

Microsoft System Journal

ISSN 0933-9434 | Januar/Februar 1989 | 3. Jg./Heft 1

öS 150,- | DM 19,80 | sfr 19,80

MS OS/2 1.1:

Große Aufgaben optimal erledigen

Serienstart:

SAA-kompatible Benutzeroberfläche in Microsoft C

Windows:

Leistungssteigerung durch Expanded Memory

Sprachen:

Das neue COBOL 3.0

C-Deklarationen verständlich gemacht

String- und Zeichenroutinen in C und Assembler

Betriebssysteme + Utilities



Für Eilbestellungen: Hotline 0 21 91-34 20 77

Ausgabe Januar/Februar 1989




Der Einsatz mehrerer Threads unter dem Presentation Manager:

Multi-Thread-Anwendungen unter MS OS/2 1.1

Microsoft Windows ist eine Multitasking-Umgebung - allerdings nicht ganz. Wie die meisten Windows-Programmierer wissen, ist Windows eine nicht preemptive Multitasking-Umgebung. Es führt kein preemptives Time-Slicing durch, das normalerweise mit einem Multitasking-System verbunden wird. Statt dessen verläßt sich Windows beim Multitasking auf das Vorhandensein von Meldungen (wobei es sich oft um Tastatur- und Mauseingaben handelt) in der Meldungs-Queue des Programms.

Wenn ein Windows-Programm die Funktion `GetMessage` aufruft, um die nächste Meldung aus der Meldungsqueue zu lesen, und diese leer ist, unterbricht Windows das Programm. Windows schaltet dann auf ein anderes Programm um, dessen Meldungsqueue nicht leer ist. Dies bewirkt, daß das andere Programm von seinem eigenen `GetMessage`-Aufruf zurückkehrt um die Meldung zu bearbeiten. Zu einem Zeitpunkt läuft immer nur ein Windows-Programm. Die anderen werden in der `GetMessage`-Funktion zeitweilig angehalten.

Windows-Programmierer sind sich der Probleme, die sich aus dieser Art des Multitasking ergeben, wohl bewußt. Wenn ein Windows-Programm zur Bearbeitung einer Meldung viel Zeit benötigt, werden andere Programme unter Windows für diese Zeit effektiv angehalten. Windows-Programmierer müssen bei der Bearbeitung langwieriger Aufgaben besondere Techniken verwenden (die ich später noch erläutern werde), um zu verhindern, daß ein Programm den Rest des Systems anhält.

Der MS OS/2 Presentation Manager ist, ähnlich wie Microsoft Windows, eine Fensterumgebung mit einer meldungsorientierten Architektur. Doch anders als Windows läuft der Presentation Manager unter einem prioritätsgesteuerten, preemptiven Multitasking-Betriebssystem.

Auf den ersten Blick könnte man glauben, daß durch das preemptive Multitasking des OS/2-Systems die Probleme, die durch die nicht preemptive Natur von Windows entstehen, beseitigt wären. Sie könnten den Schluß ziehen, daß Presentation Manager-Programme mit der Verarbeitung einer Meldung beschäftigt sein dürfen, so lange sie wollen. Doch das ist nicht so. Die Probleme, die auftreten, wenn zeitaufwendige Arbeiten unter Windows erledigt werden, ergeben sich mehr aus der meldundgsorientierten Architektur als aus dem nicht preemptiven Multitasking. In diesem Sinne verhält sich der Presentation Manager wie Windows.

Der wirkliche Unterschied liegt darin, daß der Presentation Manager mehr und bessere Lösungen für die Probleme von zeitaufwendigen Aufgaben bietet. Das soll in diesem Artikel genauer untersucht werden.

BIGJOB.H: Header-Datei

```
#define ID_RESOURCE 1

#define IDM_REPS     1
#define IDM_ACTION   2

#define IDM_10       10
#define IDM_100      100
#define IDM_1000     1000
#define IDM_10000    10000

#define IDM_START    20
#define IDM_ABORT    21

     /*-----------------------------------------------------
        Definitions, functions, and variables for BIGJOBx.C
       -----------------------------------------------------*/

#ifndef RC_INVOKED       /* This stuff not needed for .RC file */

#define STATUS_READY     0
#define STATUS_WORKING   1
#define STATUS_DONE      2

ULONG EXPENTRY ClientWndProc (HWND, USHORT, ULONG, ULONG) ;

HAB  hab ;

double Savage (double A)
     {
     return tan (atan (exp (log (sqrt (A * A))))) + 1.0 ;
     }

VOID CheckMenuItem (HWND hwnd, SHORT iMenuItem, BOOL bCheck)
     {
     HWND hwndParent = WinQueryWindow (hwnd, QW_PARENT, FALSE);
     HWND hwndMenu   = WinWindowFromID (hwndParent, FID_MENU);

     WinSendMsg (hwndMenu, MM_SETITEMATTR,
                           MAKEULONG (iMenuItem, TRUE),
                 MAKEULONG (MIA_CHECKED,
                            bCheck ? MIA_CHECKED : 0)) ;
     }

VOID EnableMenuItem (HWND hwnd, SHORT iMenuItem, BOOL bEnable)
     {
     HWND hwndParent = WinQueryWindow (hwnd, QW_PARENT, FALSE);
     HWND hwndMenu   = WinWindowFromID (hwndParent, FID_MENU);

     WinSendMsg (hwndMenu, MM_SETITEMATTR,
                           MAKEULONG (iMenuItem, TRUE),
                 MAKEULONG (MIA_DISABLED,
                            bEnable ? 0 : MIA_DISABLED)) ;
     }

VOID PaintWindow (HWND hwnd, SHORT iStatus, SHORT iRep,
                  LONG lTime)
     {
     static CHAR *szMessage [3] = { "Ready", "Working ...",
                                   "%d repetitions in %ld msec." } ;
     CHAR        szBuffer [60] ;
     HPS         hps ;
     WRECT       wrc ;

     hps = WinBeginPaint (hwnd, NULL, NULL) ;
     GpiErase (hps) ;

     WinQueryWindowRect (hwnd, &wrc) ;

     sprintf (szBuffer, szMessage [iStatus], iRep, lTime) ;

     WinDrawText (hps, -1, szBuffer, &wrc,
                                   DT_CENTER | DT_VCENTER) ;
     WinEndPaint (hps) ;
     }

#endif
```

Listing 1

BIGJOB.RC: Ressourcen-Datei

```
#include <os2.h>
#include "bigjob.h"

MENU ID_RESOURCE
     {
     SUBMENU "~Repetitions",  IDM_REPS
          {
          MENUITEM "    10",  IDM_10,    MIA_CHECKED
          MENUITEM "   100",  IDM_100
          MENUITEM "  1000",  IDM_1000
          MENUITEM " 10000",  IDM_10000
          }
     SUBMENU "~Action",       IDM_ACTION
          {
          MENUITEM "~Start",  IDM_START
          MENUITEM "~Abort",  IDM_ABORT, MIA_DISABLED
          }
     }
```

Listing 1 *(Ende)*

Das Problem der »großen Aufgabe«

Presentation Manager-Programme können Tastatur- und Mauseingaben gewöhnlich sehr schnell bearbeiten. Beispielsweise braucht in einer Textverarbeitung ein eingegebenes Zeichen nur in den Text eingefügt und auf dem Bildschirm angezeigt werden. Doch viele Programme müssen auch Befehle ausführen, die zu einer längeren Verarbeitung führen. Mein Ausdruck für diese zeitaufwendigen Aufgaben lautet »große Aufgabe«.

In einer Presentation Manager-Tabellenkalkulation ist eine große Aufgabe zum Beispiel die Neuberechnung oder die Ausführung eines langen Makros. In einem Datenbankprogramm ist die Sortierung oder die Indizierung eine große Aufgabe. In einem Textverarbeitungsprogramm ist es der Umbruch oder die Rechtschreibprüfung. In einem CAD-Programm ist es das Neuzeichnen des Bildschirms. In einem Kommunikationsprogramm das Lesen der seriellen Schnittstelle, wenn ein Zeichen nicht sofort verfügbar ist. In fast jedem Presentation Manager-Programm ist der Druck eine große Aufgabe.

Alles was länger dauert als 1/10 Sekunde, ist eine große Aufgabe. Diese Angabe stützt sich auf die Empfehlung in der Presentation Manager-Dokumentation, daß Programme zur Bearbeitung einer Meldung nicht mehr als 1/10 Sekunde benötigen sollten. Auch wenn es sich dabei nur um einen Richtwert handelt und nicht um eine unumstößliche Regel, nenne ich das die »1/10-Sekunden-Regel«.

Zur genaueren Untersuchung der Probleme bei großen Aufgaben wollen wir ein Presentation Manager-Programm schreiben, daß eine große Aufgabe erledigt, und zwar eine Berechnung, die Savage-Benchmark genannt wird, der oft für den Test der Geschwindigkeit von Fließkomma-Arithmetik verwendet wird. Das Programm erlaubt die Wiederholung der Savage-Berechnung 10, 100, 1000 oder 10000 mal, abhängig von einer Menüauswahl.

BIGJOB1: Make-Datei

```
bigjob1.obj : bigjob1.c bigjob.h
     cl -c -FPa -G2sw -W3 -Zp bigjob1.c

bigjob.res : bigjob.rc bigjob.h
     rc -r bigjob.rc

bigjob1.exe : bigjob1.obj bigjob1.def bigjob.res
     link bigjob1, /align:16, /map,
          /nod slibc slibcp slibfa os2, bigjob1
     rc bigjob.res bigjob1.exe
```

BIGJOB1.DEF: Modul-Definitionsdatei

```
NAME           BIGJOB1
DESCRIPTION    'BIGJOB Demo Program No. 1 (C) C. Petzold, 1988'
HEAPSIZE       1024
STACKSIZE      8192
EXPORTS        ClientWndProc
```

BIGJOB1.C: Eine naive Lösung für eine große Aufgabe

```
#define INCL_WIN

#include <os2.h>
#include <math.h>
#include <stdio.h>
#include "bigjob.h"

INT main (VOID)
     {
     static CHAR szClassName [] = "BigJob1" ;
     HMQ         hmq ;
     HWND        hwndFrame, hwndClient ;
     QMSG        qmsg ;

     hab = WinInitialize (0) ;
     hmq = WinCreateMsgQueue (hab, 0) ;

     WinRegisterClass (hab, szClassName, ClientWndProc,
                        CS_SYNCPAINT | CS_SIZEREDRAW, 0, NULL) ;

     hwndFrame = WinCreateStdWindow (HWND_DESKTOP,
                     WS_VISIBLE | FS_SIZEBORDER | FS_TITLEBAR
                     | FS_SYSMENU   | FS_MINMAX   | FS_MENU,
                     szClassName, "BIGJOB Demo No. 1",
                     0L, NULL, ID_RESOURCE, &hwndClient) ;

     while (WinGetMsg (hab, &qmsg, NULL, 0, 0))
          WinDispatchMsg (hab, &qmsg) ;

     WinDestroyWindow (hwndFrame) ;
     WinDestroyMsgQueue (hmq) ;
     WinTerminate (hab) ;

     return 0 ;
     }

ULONG EXPENTRY ClientWndProc (HWND hwnd, USHORT msg, ULONG mp1,
                                                     ULONG mp2)
     {
     static SHORT iCalcRep, iCurrentRep = IDM_10 ;
     static SHORT iStatus = STATUS_READY ;
     static ULONG lElapsedTime ;
     double       A ;
     SHORT        i ;

     switch (msg)
          {
```

Listing 2

```
          case WM_COMMAND:
               switch (LOUSHORT (mp1))
                    {
                    case IDM_10:
                    case IDM_100:
                    case IDM_1000:
                    case IDM_10000:
                      CheckMenuItem (hwnd, iCurrentRep, FALSE);
                      iCurrentRep = LOUSHORT (mp1) ;
                      CheckMenuItem (hwnd, iCurrentRep, TRUE) ;
                      break ;

                    case IDM_START:
                      EnableMenuItem (hwnd, IDM_START, FALSE) ;
                      EnableMenuItem (hwnd, IDM_ABORT, TRUE) ;

                      iStatus = STATUS_WORKING ;
                      WinInvalidateRect (hwnd, NULL) ;

                      iCalcRep = iCurrentRep ;
                      lElapsedTime = WinGetCurrentTime (hab) ;

                      for (A = 1.0, i = 0 ; i < iCalcRep ; i++)
                           A = Savage (A) ;

                      lElapsedTime = WinGetCurrentTime (hab) -
                                          lElapsedTime ;

                      iStatus = STATUS_DONE ;
                      WinInvalidateRect (hwnd, NULL) ;

                      EnableMenuItem (hwnd, IDM_START, TRUE) ;
                      EnableMenuItem (hwnd, IDM_ABORT, FALSE) ;
                      break ;

                    case IDM_ABORT:

                              /* Not much we can do here */

                         break ;

                    default:
                         break ;
                    }
               break ;

          case WM_PAINT:
               PaintWindow (hwnd, iStatus, iCalcRep,
                              lElapsedTime) ;
               break ;

          default:
               return WinDefWindowProc (hwnd, msg, mp1, mp2) ;
          }
     return 0L ;
     }
```

Listing 2 *(Ende)*

Ich werde für dieses Programm die alternative Fließkomma-Bibliothek verwenden, damit die Berechnung nicht durch einen eventuell vorhandenen 80287-Koprozessor beeinflußt wird. Das Programm wird über Menüoptionen verfügen, die den Start und (wenn möglich) den Abbruch der Berechnung vor ihrem Ende erlauben. Die 10000 Durchläufe des Savage-Benchmarks benötigen auf einem 8-MHz-AT etwa 3 Minuten. Schon 10 Durchläufe verletzen die 1/10-Sekunden-Regel.

Wir werden fünf Versionen dieses Programms untersuchen, wobei jede neue Version die Multitasking-Möglichkeiten des Presentation Managers ausgefeilter nutzen wird.

Die naive Vorgehensweise

Der einfachste Weg zur Erledigung einer großen Aufgabe wird im Programm BIGJOB1 in den *Listings 1 und 2* gezeigt. Der größte Teil des Codes in der Arbeitsfensterprozedur von BIGJOB1, die `ClientWndProc` genannt wird, behandelt `WM_COMMAND`-Meldungen aus dem Menü des Programms. Wenn eine Option aus dem Menü `Repetitions` (Durchläufe) gewählt wird, nimmt BIGJOB1 die Markierung der aktuellen Option weg und markiert die gewählte Option. Wenn Sie aus dem Menü `Start` wählen, gibt BIGJOB1 die Option `Abort` (Abbruch) frei und beginnt mit der Berechnung. Nachdem es die große Aufgabe erledigt hat, gibt es die `Start`-Option wieder frei und beendet die Fensterprozedur.

Da BIGJOB1 die ganze Berechnung als Reaktion auf eine `WM_COMMAND`-Meldung durchführt, verletzt es ganz klar die 1/10-Sekunden-Regel. BIGJOB1 ist ein schlechtes Programm. Ich bringe es hier nur deshalb, um zu demonstrieren, wie ein solches Programm den Rest des Presentation Managers blockiert.

Während BIGJOB1 seine große Aufgabe erledigt, können Sie nicht zu einem anderen Programm unter dem Presentation Manager umschalten. Das System ignoriert scheinbar alle Tastatur- und Mauseingaben, bis die Berechnung durchgeführt worden ist. Obwohl die Option `Abort` im Menü von BIGJOB1 vorhanden ist, können Sie weder die Tastatur noch die Maus verwenden, um diese Option auszuwählen. Sobald Sie mit der großen Aufgabe beginnen, müssen Sie warten, bis sie beendet ist, bevor Sie etwas anderes machen können.

Auf den ersten Blick scheint das sehr beunruhigend zu sein. Ist der Presentation Manager denn kein Multitasking-System? Und wenn er das ist, warum kann ein Programm ganz offensichtlich das ganze System blockieren? Was hier passiert ist das vorhersagbare Ergebnis der meldungsorientierten Architektur des Presentation Managers.

Die Verwendung von Meldungen

Programme arbeiten unter dem Presentation Manager meldungsgesteuert. Im Normalfall verbringt ein Programm die meiste Zeit in `WinGetMsg` (die Presentation Manager-Version der `GetMessage`-Funktion von Windows) und wartet auf eine Meldung.

Jedes Fenster, das von einem Programm angelegt wird, hat eine Fensterprozedur, die die Meldungen für dieses Fenster bearbeitet. Mindestens eine solche Fensterprozedur, und zwar die Fensterprozedur für die Arbeitsfläche, befindet sich im Presentation Manager-Programm. Die anderen Fensterprozeduren (zum Beispiel die für den Rahmen, die Titelleiste und das Menü) befinden sich in der dynamischen Link-Library PMWIND.DLL des Presentation Managers.

BIGJOB2: Make-Datei

```
bigjob2.obj : bigjob2.c bigjob.h
     cl -c -FPa -G2sw -W3 -Zp bigjob2.c

bigjob.res : bigjob.rc bigjob.h
     rc -r bigjob.rc

bigjob2.exe : bigjob2.obj bigjob2.def bigjob.res
     link bigjob2, /align:16, /map,
          /nod slibc slibcp slibfa os2, bigjob2
     rc bigjob.res bigjob2.exe
```

BIGJOB2.DEF: Modul-Definitionsdatei

```
NAME            BIGJOB2
DESCRIPTION     'BIGJOB Demo Program No. 2 (C) C. Petzold, 1988'
HEAPSIZE        1024
STACKSIZE       8192
EXPORTS         ClientWndProc
```

BIGJOB2.C: Timer-Lösung für eine große Aufgabe

```
#define INCL_WIN

#include <os2.h>
#include <math.h>
#include <stdio.h>
#include "bigjob.h"

#define ID_TIMER 1

INT main (VOID)
     {
     static CHAR szClassName [] = "BigJob2" ;
     HMQ         hmq ;
     HWND        hwndFrame, hwndClient ;
     QMSG        qmsg ;

     hab = WinInitialize (0) ;
     hmq = WinCreateMsgQueue (hab, 0) ;

     WinRegisterClass (hab, szClassName, ClientWndProc,
                       CS_SYNCPAINT | CS_SIZEREDRAW, 0, NULL) ;

     hwndFrame = WinCreateStdWindow (HWND_DESKTOP,
                    WS_VISIBLE | FS_SIZEBORDER | FS_TITLEBAR
                    | FS_SYSMENU    | FS_MINMAX   | FS_MENU,
                    szClassName, "BigJob Demo No. 2",
                    0L, NULL, ID_RESOURCE, &hwndClient) ;

     while (WinGetMsg (hab, &qmsg, NULL, 0, 0))
          WinDispatchMsg (hab, &qmsg) ;

     WinDestroyWindow (hwndFrame) ;
     WinDestroyMsgQueue (hmq) ;
     WinTerminate (hab) ;

     return 0 ;
     }

ULONG EXPENTRY ClientWndProc (HWND hwnd, USHORT msg, ULONG mp1,
                                                     ULONG mp2)
     {
     static double A ;
     static SHORT  i, iCalcRep, iCurrentRep = IDM_10 ;
     static SHORT  iStatus = STATUS_READY ;
     static ULONG  lElapsedTime ;

     switch (msg)
          {
```

Listing 3

```
          case WM_COMMAND:
               switch (LOUSHORT (mp1))
                    {
                    case IDM_10:
                    case IDM_100:
                    case IDM_1000:
                    case IDM_10000:
                         CheckMenuItem (hwnd, iCurrentRep, FALSE);
                         iCurrentRep = LOUSHORT (mp1) ;
                         CheckMenuItem (hwnd, iCurrentRep, TRUE) ;
                         break ;

                    case IDM_START:

                         if (!WinStartTimer (hab, hwnd,
                                             ID_TIMER, 1))
                              {
                              WinAlarm (HWND_DESKTOP,WA_ERROR);
                              break ;
                              }

                         EnableMenuItem (hwnd,IDM_START,FALSE);
                         EnableMenuItem (hwnd,IDM_ABORT,TRUE);

                         iStatus = STATUS_WORKING ;
                         WinInvalidateRect (hwnd, NULL) ;

                         iCalcRep = iCurrentRep ;
                         lElapsedTime = WinGetCurrentTime(hab);

                         A = 1.0 ;
                         i = 0 ;

                         break ;

                    case IDM_ABORT:
                         WinStopTimer (hab, hwnd, ID_TIMER) ;

                         iStatus = STATUS_READY ;
                         WinInvalidateRect (hwnd, NULL) ;

                         EnableMenuItem (hwnd,IDM_START,TRUE);
                         EnableMenuItem (hwnd,IDM_ABORT,FALSE);
                         break ;

                    default:
                         break ;
                    }
               break ;

          case WM_TIMER:

               A = Savage (A) ;

               if (++i == iCalcRep)
                    {
                    lElapsedTime = WinGetCurrentTime (hab) -
                                        lElapsedTime ;

                    WinStopTimer (hab, hwnd, ID_TIMER) ;

                    iStatus = STATUS_DONE ;
                    WinInvalidateRect (hwnd, NULL) ;

                    EnableMenuItem (hwnd, IDM_START, TRUE) ;
                    EnableMenuItem (hwnd, IDM_ABORT, FALSE) ;
                    }
               break ;

          case WM_PAINT:
               PaintWindow(hwnd,iStatus,iCalcRep,lElapsedTime);
               break ;

          default:
               return WinDefWindowProc (hwnd, msg, mp1, mp2) ;
          }
     return 0L ;
     }
```

Listing 3 *(Ende)*

Einige der Meldungen für ein Fenster werden in der Meldungsqueue des Programms gespeichert. Diese Meldungen werden »gequeute« Meldungen genannt und man sagt, daß sie in die Queue »eingetragen« wurden. Die gequeueten Meldungen werden aus der Meldungsqueue gelesen, wenn ein Programm `WinGetMsg` aufruft und werden mit `WinDispatchMsg` an die Fensterprozedur weitergeleitet. Andere Meldungen werden direkt, unter Umgehung der Meldungsqueue, an die Fensterprozedur geschickt.

Ganz gleich, ob eine Meldung in die Meldungsqueue eingetragen oder direkt an die Fensterprozedur geschickt wird und ganz gleich, ob sich die Fensterprozedur im Programm oder in einer dynamischen Link-Library befindet, Meldungen an Fenster, die von einem Ausführungsthread erzeugt werden, werden immer in diesem Thread verarbeitet. Ein bestimmter Thread kann zu einem Zeitpunkt nur eine Aufgabe ausführen. Ein Thread kann nicht mit sich selbst im Multitasking-Betrieb laufen. Das scheint ganz klar zu sein, doch die meldungsorientierte Architektur des Presentation Managers verschleiert diese simple Tatsache.

Wenn Sie aus dem Menü von BIGJOB1 `Start` wählen, beginnt das Programm mit der großen Aufgabe. Sie versuchen dann, mit der Tastenkombination [Alt][Tab] zu einem anderen Programm umzuschalten. Das Fenster, daß diese Tastendrücke verarbeiten muß, ist das Rahmenfenster von BIGJOB1. Die Fensterprozedur für das Rahmenfenster muß im selben Thread laufen wie das Arbeitsfenster, doch das Arbeitsfenster ist gerade emsig damit beschäftigt, die große Aufgabe zu erledigen. Das bedeutet, daß die Tastaturmeldung [Alt][Tab] nicht eher bearbeitet werden kann, als bis die Berechnung beendet ist, BIGJOB1 `ClientWndProc` verläßt und dann `WinGetMsg` aufruft, um die Meldung aus der Queue zu lesen. Das erklärt, warum der Presentation Manager Tastatureingaben scheinbar ignoriert, während BIGJOB1 rechnet.

Serialisierung der Eingabe

Es scheint jedoch einen Weg zu geben, dies zu umgehen. Wie Sie wissen, kann man ein anderes Fenster auch mit der Maus aktivieren. Vielleicht funktioniert das.

Um diese Möglichkeit zu prüfen, müssen Sie den Mauszeiger zunächst im Fenster eines anderen Programms positionieren und dann über die Tastatur `Start` aus dem Menü von BIGJOB auswählen. Während das Programm rechnet drücken Sie die Maustaste und ... nichts passiert.

Auch dies ist zunächst wieder etwas beunruhigend. Da der Presentation Manager ein echtes Multitasking-System ist, sollte das andere Programm den Mausklick lesen können, selbst wenn BIGJOB1 rechnet. Der Presentation Manager sollte es ebenso erlauben, daß ein anderes Programm aktiviert wird.

Das passiert aus einer Reihe von Gründen jedoch nicht. Zum einen serialisiert der Presentation Manager alle Tastatur- und Mauseingaben in einer System-Meldungsqueue. Die Eingaben werden Meldung für Meldung an die Meldungsqueue einer Anwendung übergeben, abhängig davon, welches Fenster den Eingabefokus hat (bei der Tastatur) und welches Fenster sich unter dem Mauszeiger befindet.

Die Serialisierung von Maus- und Tastatureingaben in einer System-Meldungsqueue ist notwendig, um die Pufferung dieser Benutzereingaben korrekt handhaben zu können. Einer der Tastendrücke oder Mausklicks in der System-Meldungsqueue könnte bewirken, daß sich das aktive Fenster und das Fokusfenster ändern. Darauffolgende Tastatureingaben müssen an das neue Fenster gehen. Dies würde nicht funktionieren, wenn Maus- und Tastatureingaben nicht in derselben Reihenfolge an Anwendungen übergeben würden, in der sie in die System-Meldungsqueue eingetragen werden. Deshalb kann eine Tastatur- oder Mausmeldung nicht eher an eine bestimmte Anwendung übergeben werden, bis alle vorherigen Tastatur- und Mausmeldungen verarbeitet worden sind.

In diesem speziellen Fall können andere Anwendungen solange keine Mausmeldung lesen, bis BIGJOB1 alle seine Tastatureingaben verarbeitet hat. Bei der Tastatureingabe, die noch nicht verarbeitet wurde, handelt es sich um das Loslassen der Taste, das bewirkte, daß das Menü die Meldung `WM_COMMAND` geschickt hat, mit der die Berechnung gestartet wurde.

So wiedersetzt sich BIGJOB1 nicht nur Tastatur- und Mauseingaben, es hindert auch noch alle anderen Programme, die gerade unter dem Presentation Manager laufen, daran, Tastatur- oder Mauseingaben zu erhalten.

Selbst wenn ein anderes Programm einen Mausklick lesen könnte, kann der Presentation Manager den Eingabefokus nicht von BIGJOB1 an ein anderes Programm abgeben, während BIGJOB1 damit beschäftigt ist, Berechnungen durchzuführen. Um den Eingabefokus zu ändern, muß der Presentation Manager die Meldung `WM_SETFOCUS` an das Fenster schicken, das den Eingabefokus verliert. Diese `WM_SETFOCUS`-Meldung wird blockiert, weil das Fenster, das die Meldung erhalten muß, ein Teil des BIGJOB1-Threads ist und BIGJOB1 ist damit beschäftigt, seine große Aufgabe zu erledigen.

Meldungen sind etwas anderes als Hardware-Interrupts. Auch wenn einer Fensterprozedur als Ergebnis eines `WinDefWindowProc`-Aufrufs und einer Fensterprozedur als Ergebnis anderer Presentation Manager-Funktionen eine Meldung geschickt werden kann, sind dies Beispiele für die Rekursion in Fensterprozeduren. Meldungen unterbrechen nicht preemptiv einen Thread und setzen die Ausführung irgendwo anders in dem gleichen Thread fort.

Nachdem Sie nun gesehen haben, wie BIGJOB1 die Tastatur- und Mauseingaben im Presentation Manager effektiv ausschaltet, sollte der Grund für die 1/10-Sekunden-Regel klar sein. Presentation Manager-Programme müssen ständig mit dem System kommunizieren und ihre Meldungen zügig lesen und verarbeiten.

BIGJOB3: Make-Datei

```
bigjob3.obj : bigjob3.c bigjob.h
     cl -c -FPa -G2sw -W3 -Zp bigjob3.c

bigjob.res : bigjob.rc bigjob.h
     rc -r bigjob.rc

bigjob3.exe : bigjob3.obj bigjob3.def bigjob.res
     link bigjob3, /align:16, /map,
          /nod slibc slibcp slibfa os2, bigjob3
     rc bigjob.res bigjob3.exe
```

BIGJOB3.DEF: Modul-Definitionsdatei

```
NAME            BIGJOB3
DESCRIPTION     'BigJob Demo Program No. 3 (C) C. Petzold, 1988'
HEAPSIZE        1024
STACKSIZE       8192
EXPORTS         ClientWndProc
```

BIGJOB3.C: PeekMessage für eine große Aufgabe

```
#define INCL_WIN

#include <os2.h>
#include <math.h>
#include <stdio.h>
#include "bigjob.h"

INT main (VOID)
     {
     static CHAR szClassName [] = "BigJob3" ;
     HMQ          hmq ;
     HWND         hwndFrame, hwndClient ;
     QMSG         qmsg ;

     hab = WinInitialize (0) ;
     hmq = WinCreateMsgQueue (hab, 0) ;

     WinRegisterClass (hab, szClassName, ClientWndProc,
                            CS_SIZEREDRAW, 0, NULL) ;

     hwndFrame = WinCreateStdWindow (HWND_DESKTOP,
                    WS_VISIBLE | FS_SIZEBORDER | FS_TITLEBAR
                    | FS_SYSMENU    | FS_MINMAX   | FS_MENU,
                    szClassName, "BigJob Demo No. 3",
                    0L, NULL, ID_RESOURCE, &hwndClient) ;

     while (WinGetMsg (hab, &qmsg, NULL, 0, 0))
          WinDispatchMsg (hab, &qmsg) ;

     WinDestroyWindow (hwndFrame) ;
     WinDestroyMsgQueue (hmq) ;
     WinTerminate (hab) ;

     return 0 ;
     }

ULONG EXPENTRY ClientWndProc (HWND hwnd, USHORT msg, ULONG mp1,
                                                     ULONG mp2)
     {
     static BOOL   bContinueCalc = FALSE ;
     static SHORT  iStatus = STATUS_READY ;
     static SHORT  iCalcRep, iCurrentRep = IDM_10 ;
     static ULONG  lElapsedTime ;
     double        A ;
     SHORT         i ;
     QMSG          qmsg ;
```

Listing 4

```
     switch (msg)
          {
          case WM_COMMAND:
               switch (LOUSHORT (mp1))
                    {
                    case IDM_10:
                    case IDM_100:
                    case IDM_1000:
                    case IDM_10000:
                         CheckMenuItem(hwnd,iCurrentRep,FALSE);
                         iCurrentRep = LOUSHORT (mp1) ;
                         CheckMenuItem(hwnd,iCurrentRep,TRUE);
                         break ;
                    case IDM_START:
                         EnableMenuItem(hwnd,IDM_START,FALSE);
                         EnableMenuItem(hwnd,IDM_ABORT,TRUE);

                         iStatus = STATUS_WORKING ;
                         WinInvalidateRect (hwnd, NULL) ;

                         iCalcRep = iCurrentRep ;
                         bContinueCalc = TRUE ;
                         lElapsedTime = WinGetCurrentTime(hab);

                         qmsg.msg = WM_NULL ;

                         for (A = 1.0, i = 0; i < iCalcRep;i++)
                              {
                              A = Savage (A) ;

                              while (WinPeekMsg (hab, &qmsg,
                                                 NULL, 0, 0,
                                                 PM_NOREMOVE))
                                   {
                                   if (qmsg.msg == WM_QUIT)
                                        break ;

                                   WinGetMsg (hab, &qmsg,
                                              NULL, 0, 0) ;
                                   WinDispatchMsg (hab, &qmsg);

                                   if (!bContinueCalc)
                                        break ;
                                   }

                              if (!bContinueCalc ||
                                        qmsg.msg == WM_QUIT)
                                   break ;
                              }
                         lElapsedTime = WinGetCurrentTime(hab)-
                                                  lElapsedTime ;

                         if (!bContinueCalc ||
                                   qmsg.msg == WM_QUIT)
                              iStatus = STATUS_READY ;
                         else
                              iStatus = STATUS_DONE ;

                         WinInvalidateRect (hwnd, NULL) ;

                         EnableMenuItem(hwnd,IDM_START,TRUE);
                         EnableMenuItem(hwnd,IDM_ABORT,FALSE);
                         break ;
                    case IDM_ABORT:
                         bContinueCalc = FALSE ;
                         break ;
                    default:
                         break ;
                    }
               break ;
          case WM_PAINT:
               PaintWindow(hwnd,iStatus,iCalcRep,lElapsedTime);
               break ;
          default:
               return WinDefWindowProc (hwnd, msg, mp1, mp2) ;
          }
     return 0L ;
     }
```

Listing 4 *(Ende)*

Nicht genau wie Windows

So schlecht BIGJOB1 auch sein mag, der Presentation Manager kann auch dann den Multitasking-Betrieb fortsetzen, wenn BIGJOB1 läuft. Dies ist ein Unterschied zwischen dem Presentation Manager und Windows.

Wenn Sie eine Windows-Version von BIGJOB1 unter Windows zusammen mit dem Uhr-Programm laufenlassen, bleibt die Uhr stehen, während BIGJOB1 rechnet. Unter Windows kann zu einem Zeitpunkt nur ein Programm laufen. Wenn Sie BIGJOB1 jedoch unter dem Presentation Manager zusammen mit dessen Version der Uhr laufenlassen (wie zum Beispiel der aus meinem Buch *Programming the OS/2 Presentation Manager*, Microsoft Press, 1988), werden Sie sehen, daß die Uhr weiterläuft, während BIGJOB1 rechnet. Die Uhr und BIGJOB1 laufen gleichzeitig.

Dies funktioniert durch Setzen eines Timers und durch Verarbeitung einer `WM_TIMER`-Meldung in jeder Sekunde. Eine `WM_TIMER`-Meldung braucht nicht mit Tastatur- und Mauseingaben serialisiert werden. Die Uhr kann weiterhin diese Meldungen erhalten, selbst wenn BIGJOB1 Tastatur- und Mauseingaben blockiert hat.

Der Einsatz eines Timers

Wenn Sie erkannt haben, daß BIGJOB1 ein sehr schlechtes Presentation Manager-Programm ist, haben Sie das Problem, es so umzustrukturieren, daß es richtig arbeitet. Die Verwendung eines Timers bei der Uhr scheint eine Lösung aufzuzeigen: Man kann eine große Aufgabe in kleine Teile aufteilen, den Presentation Manager-Timer setzen und jeden kleinen Teil beim Empfang einer `WM_TIMER`-Meldung ausführen. Dies ist eine Lösung, die sowohl unter Windows, als auch unter dem Presentation Manager funktioniert.

Das Programm BIGJOB2 verwendet einen Timer für die große Aufgabe. Die Dateien BIGJOB2, BIGJOB2.C und BIGJOB2.DEF sind in *Listing 3* zu sehen. Für die Kompilierung von BIGJOB2 werden auch die Dateien BIGJOB.H und BIGJOB.RC aus *Listing 1* benötigt.

Wenn Sie die Option `Start` aus dem Menü von BIGJOB2 wählen, ruft BIGJOB2 `WinStartTimer` auf, um den Timer zu starten, es schaltet die `Start`-Option aus, die `Abort`-Option ein und initialisiert einige Variablen. Die Savage-Berechnung wird einmal bei jeder `WM_TIMER`-Meldung ausgeführt. Bei 1000 Durchläufen wird die große Aufgabe also nach 1000 `WM_TIMER`-Meldungen beendet.

`WM_TIMER`-Meldungen sind Meldungen mit niedriger Priorität. Wenn die Meldungsqueue Tastatur- oder Mausmeldungen enthält, werden diese Meldungen vor einer `WM_TIMER`-Meldung aus der Meldungsqueue gelesen und verarbeitet. So kann BIGJOB2 weiter Tastatur- und Mauseingaben lesen, und es dem Benutzer gleichzeitig ermöglichen, `Abort` aus dem Menü von BIGJOB2 zu wählen, das Fenster von BIGJOB2 zu bewegen oder zu vergrößern oder zu einem anderen Programm zu wechseln. Das ganze System, auch BIGJOB2 selbst, arbeitet die ganze Zeit normal, während BIGJOB2 seine Berechnungen durchführt.

Timer-Probleme

Der Timer-Ansatz ist für BIGJOB2 nicht schlecht, es gibt jedoch Fälle in denen ein Timer völlig unpassend wäre.

Ein Programm, das den Timer verwendet, muß laufend die Verarbeitungsschleife bei jeder `WM_TIMER`-Meldung betreten und wieder verlassen. Dies kann man leicht strukturieren, wenn es sich um eine Schleife handelt, es wird jedoch bei komplizierteren Aufgaben mit mehreren verschachtelten Schleifen zum Alptraum.

Darüber hinaus verlangsamt der Timer die große Aufgabe. Es ist nicht möglich, `WM_TIMER`-Meldungen öfter zu erhalten, als dies die Hardware-Uhr zuläßt. Unter OS/2 bedeutet dies, daß das Programm nur alle 31,25 Millisekunden eine `WM_TIMER`-Meldung erhält. BIGJOB2 benötigt jedoch weniger als 20 Millisekunden (auf einem 8-MHz-AT) zur Verarbeitung jeder `WM_TIMER`-Meldung. Da die Berechnung im Takt mit dem Timer erfolgt, wird die Berechnung auf einem 20-MHz-80386-System um keinen Deut schneller erledigt.

Als allgemeine Lösung für die Erledigung einer großen Aufgabe muß der Timer-Ansatz also eindeutig verworfen werden.

PeekMessage

Eine andere häufig verwendete Lösung unter Windows besteht in der Verwendung der Funktion `PeekMessage` (`WinPeekMsg` unter dem Presentation Manager). Die Funktion `PeekMessage` gleicht der Funktion `GetMessage`. Unter Windows prüft die Funktion `PeekMessage` zuerst, ob irgendwelche Meldungen in der Meldungsqueue des Programms warten. Wenn das so ist, füllt `PeekMessage` die Meldungsstruktur mit der nächsten Meldung aus der Queue und gibt einen Wert ungleich Null zurück.

Wenn die Meldungsqueue des Programms, daß `PeekMessage` aufruft, leer ist, schaltet Windows zu einem Programm um, dessen Meldungsqueue nicht leer ist, um diesem Programm die Bearbeitung seiner Meldungen zu ermöglichen. (Dies wird oft »Übergabe« an ein anderes Programm genannt.) Wenn in den Meldungsqueues der Programme keine Meldungen übrig bleiben, übergibt `PeekMessage` die Steuerung wieder an das Originalprogramm und gibt Null zurück.

`PeekMessage` ermöglicht allen Windows-Programmen die Verarbeitung offener Meldungen, gibt die Steuerung jedoch dann an das Programm zurück, das `PeekMessage` aufgerufen hat, nachdem die ganze Meldungsverarbeitung beendet ist. Die Verwendung von `PeekMessage` erlaubt es einem Programm, eine große Aufgabe zu erledigen, wäh-

rend es ihm und auch anderen Programmen periodisch möglich ist, ihre Meldungen zu verarbeiten. Mehrere Windows-Programme, einschließlich SPOOLER und TERMINAL, verwenden `PeekMessage` in dieser Art, um Multitasking zu simulieren. Programme, die drucken, verwenden häufig `PeekMessage`, damit der Benutzer den Druckvorgang mit einem Dialogfeld abbrechen kann.

Die Funktion `WinPeekMsg` des Presentation Managers gleicht `PeekMessage`. Im Presentation Manager wird `WinPeekMsg` jedoch von einem Programm für die Verarbeitung seiner eigenen Meldungen verwendet und nicht zur Übergabe der Ausführung an andere Programme. Multitasking wird unter dem Presentation Manager normalerweise dann ausgeführt, wenn ein Programm die Verarbeitung der Tastatur- und Mauseingaben nicht blockiert. Die Funktion `WinPeekMsg` bietet eine gute Lösung für das Problem der großen Aufgabe. BIGJOB3 in *Listing 4* zeigt, wie es gemacht wird.

Wie BIGJOB2 erledigt BIGJOB3 alle Berechnungen als Reaktion auf eine `WM_COMMAND`-Meldung. In der Berechnungsschleife ruft BIGJOB3 `WinPeekMsg` auf, um festzustellen, ob sich in seiner Meldungsqueue Meldungen befinden. Wenn das der Fall ist, entfernt es sie mit `WinGetMsg` und leitet sie mit `WinDisPatchMsg` an eine Fensterprozedur wie die normale Meldungsschleife in `main` weiter.

Bei einer dieser Meldungen könnte es sich um eine `WM_COMMAND`-Meldung handeln, die erzeugt wird, wenn der Benutzer aus dem Menü `Abort` wählt. BIGJOB3 verwendet `bContinueCalc` als Flag für den Abbruch.

Sie werden bemerkt haben, daß für die Meldung `WM_QUIT` eine besondere Verarbeitung erforderlich ist. Diese Meldung wird vom Presentation Manager als Standardantwort auf die Auswahl von `Close` im Systemmenü in die Meldungsqueue eingetragen. Die Meldung `WM_QUIT` sollte in der Fensterprozedur nicht aus der Queue entfernt werden. Statt dessen verläßt BIGJOB3 die Fensterprozedur, damit die `WM_QUIT`-Meldung von der `main`-Funktion ausgelesen werden kann.

Auch wenn diese Meldungs-Vorausschau in Windows- und Presentation Manager-Programmen gewöhnlich gut funktioniert, ist sie in der Praxis immer etwas unsauber. Die Funktionen `PeekMessage` und `WinPeekMsg` müssen oft genug aufgerufen werden, um ein gutes Antwortverhalten des Systems zu erzielen, und es wird eine unverhältnismäßig große Menge von Programmcode benötigt. Wenn die große Aufgabe abgebrochen werden muß, ist es gelegentlich schwierig, die Berechnungsschleife in einer strukturierten Art und Weise zu verlassen. Kurz gesagt, ein Programm mit `WinPeekMsg` sieht oft arg verunglückt aus.

Ein zweiter Thread

Die Lösungen von BIGJOB2 und BIGJOB3 können sowohl unter Windows als auch beim Presentation Manager verwendet werden. Lassen Sie uns nun etwas tun, was unter Windows unmöglich, beim Presentation Manager jedoch ganz natürlich ist: Wir wollen für die große Aufgabe einen zweiten Ausführungsthread erzeugen.

Wenn ein Programm mehrere Ausführungsthreads enthält, laufen dieses Threads gleichzeitig. Alle Threads in einem Prozeß teilen sich die Ressourcen des Programms, zum Beispiel offene Dateien, Speicher, Semaphore und Queues, doch jeder Thread verfügt über einen eigenen CPU-Zustand, eigene Zuteilungspriorität und einen eigenen Stack.

In einem Programm sieht ein zweiter Ausführungsthread wie eine Funktion aus. Alle lokalen automatischen Variablen in einer Threadfunktion sind privates Eigentum dieses Threads, denn sie werden auf dem Stack des Threads gespeichert. Lokale statische Variablen in der Threadfunktion können von allen Threads, die auf dieser Funktion basieren, gemeinsam verwendet werden.

Bei der Programmierung für den Presentation Manager fallen Threads in zwei Kategorien: Ein Thread ist entweder ein Thread mit oder ohne Meldungsqueue. Ein Thread wird zu einem Thread mit Meldungsqueue, wenn er `WinCreateMsgQueue` aufruft. Er wird wieder ein Thread ohne Meldungsqueue, wenn er `WinDestroyMsgQueue` aufruft.

Ein Presentation Manager-Programm legt immer in mindestens einem Thread eine Meldungsqueue an. Ein Thread muß eine Meldungsqueue anlegen, bevor er Fenster erzeugen kann. Die Meldungsqueue wird dazu verwendet, um die Meldungen für alle Fenster in dem Thread zu speichern. Andere Threads in einem Presentation Manager-Programm brauchen jedoch keine Meldungsqueue anlegen, wenn sie keine Fenster erzeugen.

Beschränkungen von Threads

In einem Presentation Manager-Programm haben Threads ohne Meldungsqueue gewisse Vorteile gegenüber solchen mit Meldungsqueue - aber auch einige Nachteile.

Erfreulich ist, daß ein Thread ohne Meldungsqueue nicht an die 1/10-Sekunden-Regel gebunden ist, denn er empfängt oder verarbeitet keine Meldungen, er braucht nicht zu befürchten, daß er in Threads mit Meldungsqueue die Meldungsverarbeitung blockiert. Somit eignet sich ein Thread ohne Meldungsqueue oft ideal für große Aufgaben.

Unerfreulich ist, daß Threads ohne Meldungsqueue in der Verwendbarkeit von Presentation Manager-Funktionen eingeschränkt sind. Im Prinzip kann ein Thread ohne eine Meldungsqueue keine Fenster anlegen, keine Meldungen an einen Thread mit Meldungsqueue schicken und keine Funktionen aufrufen, die dazu führen, daß Meldungen an eine Fensterfunktion geschickt werden.

Einige dieser Beschränkungen sind sofort einsichtig: Ein Thread ohne Meldungsqueue kann kein Fenster anlegen, weil er keine Queue hat, in der er Meldungen für das Fenster speichern könnte.

BIGJOB4: Make-Datei

```
bigjob4.obj : bigjob4.c bigjob.h
     cl -c -FPa -G2sw -W3 -Zp bigjob4.c

bigjob.res : bigjob.rc bigjob.h
     rc -r bigjob.rc

bigjob4.exe : bigjob4.obj bigjob4.def bigjob.res
     link bigjob4, /align:16, /map,
          /nod slibc slibcp slibfa os2, bigjob4
     rc bigjob.res bigjob4.exe
```

BIGJOB4.DEF: Modul-Definitionsdatei

```
NAME           BIGJOB4
DESCRIPTION    'BigJob Demo Program No. 4 (C) C. Petzold, 1988'
HEAPSIZE       1024
STACKSIZE      8192
EXPORTS        ClientWndProc
```

BIGJOB4.C: Ein zweiter Thread für die große Aufgabe

```
#define INCL_WIN
#define INCL_DOS

#include <os2.h>
#include <math.h>
#include <stdio.h>
#include "bigjob.h"

#define WM_CALC_DONE      (WM_USER + 0)
#define WM_CALC_ABORTED   (WM_USER + 1)

VOID FAR SecondThread (VOID) ;

BOOL  bContinueCalc = FALSE ;
HWND  hwndClient ;
SHORT iCalcRep ;
LONG  lSemTrigger ;
TID   idThread ;
UCHAR cThreadStack [4096] ;

INT main (VOID)
     {
     static CHAR szClassName [] = "BigJob4" ;
     HMQ          hmq ;
     HWND         hwndFrame ;
     QMSG         qmsg ;

     hab = WinInitialize (0) ;
     hmq = WinCreateMsgQueue (hab, 0) ;

     WinRegisterClass (hab, szClassName, ClientWndProc,
                                CS_SIZEREDRAW, 0, NULL) ;

     hwndFrame = WinCreateStdWindow (HWND_DESKTOP,
                    WS_VISIBLE | FS_SIZEBORDER | FS_TITLEBAR
                    | FS_SYSMENU     | FS_MINMAX    | FS_MENU,
                    szClassName, "BigJob Demo No. 4",
                    0L, NULL, ID_RESOURCE, &hwndClient) ;

     while (WinGetMsg (hab, &qmsg, NULL, 0, 0))
          WinDispatchMsg (hab, &qmsg) ;
     DosSuspendThread (idThread) ;
     WinDestroyWindow (hwndFrame) ;
     WinDestroyMsgQueue (hmq) ;
     WinTerminate (hab) ;
     return 0 ;
     }
```

Listing 5

```
ULONG EXPENTRY ClientWndProc (HWND hwnd, USHORT msg, ULONG mp1,
                                                     ULONG mp2)
     {
     static SHORT iCurrentRep = IDM_10 ;
     static SHORT iStatus = STATUS_READY ;
     static ULONG lElapsedTime ;

     switch (msg)
          {
          case WM_CREATE:
               DosSemSet (&lSemTrigger) ;

               if (DosCreateThread (SecondThread, &idThread,
                         cThreadStack + sizeof cThreadStack))

                    WinAlarm (HWND_DESKTOP, WA_ERROR) ;

               break ;

          case WM_COMMAND:
               switch (LOUSHORT (mp1))
                    {
                    case IDM_10:
                    case IDM_100:
                    case IDM_1000:
                    case IDM_10000:
                         CheckMenuItem(hwnd,iCurrentRep,FALSE);
                         iCurrentRep = LOUSHORT (mp1) ;
                         CheckMenuItem(hwnd,iCurrentRep,TRUE);
                         break ;

                    case IDM_START:
                         iStatus = STATUS_WORKING ;
                         WinInvalidateRect (hwnd, NULL) ;

                         iCalcRep = iCurrentRep ;
                         bContinueCalc = TRUE ;
                         DosSemClear (&lSemTrigger) ;

                         EnableMenuItem(hwnd,IDM_START,FALSE);
                         EnableMenuItem(hwnd,IDM_ABORT,TRUE);
                         break ;

                    case IDM_ABORT:
                         bContinueCalc = FALSE ;

                         EnableMenuItem(hwnd,IDM_ABORT,FALSE);
                         break ;

                    default:
                         break ;
                    }
               break ;

          case WM_CALC_DONE:
               iStatus = STATUS_DONE ;
               lElapsedTime = mp1 ;
               WinInvalidateRect (hwnd, NULL) ;

               EnableMenuItem (hwnd, IDM_START, TRUE) ;
               EnableMenuItem (hwnd, IDM_ABORT, FALSE) ;
               break ;

          case WM_CALC_ABORTED:
               iStatus = STATUS_READY ;
               WinInvalidateRect (hwnd, NULL) ;

               EnableMenuItem (hwnd, IDM_START, TRUE) ;
               break ;
          case WM_PAINT:
               PaintWindow(hwnd,iStatus,iCalcRep,lElapsedTime);
               break ;
          default:
               return WinDefWindowProc (hwnd, msg, mp1, mp2) ;
          }
     return 0L ;
     }
```

Listing 5 *(Fortsetzung)*

```
VOID FAR SecondThread ()
     {
     double A ;
     int    i ;
     LONG   lTime ;

     while (1)
          {
          DosSemWait (&lSemTrigger, -1L) ;

          lTime = WinGetCurrentTime (hab) ;

          for (A = 1.0, i = 0 ; i < iCalcRep ; i++)
               {
               if (!bContinueCalc)
                    break ;

               A = Savage (A) ;
               }

          lTime = WinGetCurrentTime (hab) - lTime ;

          DosSemSet (&lSemTrigger) ;

          if (bContinueCalc)
               WinPostMsg(hwndClient, WM_CALC_DONE, lTime, 0L);
          else
               WinPostMsg(hwndClient, WM_CALC_ABORTED, 0L, 0L);
          }
     }
```

Listing 5 *(Ende)*

Ein Thread ohne Meldungsqueue kann jedoch einige Funktionen aufrufen, die Fenster in einem Thread mit Meldungsqueue beeinflussen. Beispielsweise kann ein Thread ohne Meldungsqueue eine Handle auf einen Präsentationsbereich in einem Fenster anfordern, das von einem Thread mit Meldungsqueue angelegt wurde, und dadurch etwas in diesem Fenster anzeigen.

Threads ohne Meldungsqueue können keine Meldungen an Thread mit Meldungsqueue schicken. Die Funktion `WinSendMsg` darf nicht verwendet werden. Auch können sie keine Funktionen aufrufen, die Meldungen schicken. Die Funktion `WinDestroyWindows` kann zum Beispiel von einem Thread ohne Meldungsqueue nicht aufgerufen werden, weil sie die Meldung `WM_DESTROY` an eine Fensterfunktion schickt. Die Funktionen, die für Threads ohne Meldungsqueue verboten sind, sind in der Programmierdokumentation des Presentation Managers aufgeführt.

Obwohl ein Thread ohne Meldungsqueue keine Meldungen mit `WinSendMsg` schicken kann, kann der Thread eine Meldung mit `WinPostMsg` eintragen. Diese Funktion schreibt die Meldung in die Meldungsqueue eines Threads und kehrt sofort zurück.

Semaphore

Mehrere Threads in einem einzigen Prozeß müssen oft miteinander auf viele Arten kommunizieren. Die Ausführung von Threads muß koordiniert werden, damit sie sich nicht gegenseitig auf die Füße treten. Dies erfordert einige Abstimmung. Oft müssen Threads sich gegenseitig auch etwas signalisieren oder Daten übergeben.

Ein Thread mit Meldungsqueue kommuniziert mit einem Thread ohne Meldungsqueue in der Regel über Semaphore. Ein Thread ohne Meldungsqueue kommuniziert mit einem Thread mit Meldungsqueue durch das Eintragen von Meldungen. Beide Threads können auch auf gemeinsame globale Variablen zugreifen.

Das Programm BIGJOB4 in *Listing 5*, das einen Thread ohne Meldungsqueue für die große Aufgabe einsetzt, demonstriert diese Kommunikation.

Die globale Varibale `lSemTrigger` ist ein RAM-Semaphor. Dieses Semaphor wird bei der Meldung `WM_CREATE` in der Prozedur `ClientWndProc` gesetzt. Die Fensterprozedur erzeugt dann mit einem Aufruf der OS/2-Funktion `DosCreateThread` einen zweiten Thread, wobei `SecondThread` die Threadfunktion ist.

`SecondThread` wartet darauf, daß `lSemTrigger` zurückgesetzt wird. `ClientWndProc` setzt das Semaphor zurück, wenn der Benutzer aus dem Menü von BIGJOB4 `Start` auswählt. `SecondThread` führt dann die Berechnungen aus. Wenn er fertig ist, übergibt er die Meldung `WM_CALC_DONE` an die Fensterprozedur. (Dabei handelt es sich um eine benutzerdefinierte Meldung.) Die für die Berechnung benötigte Zeit wird der Fensterprozedur in der Variablen `lParam1` übergeben, die ein Bestandteil dieser Meldung ist.

Abbruch der Berechnung

BIGJOB4 erlaubt es dem Benutzer auch, die Berechnung abzubrechen. Wenn aus dem Menü `Abort` gewählt wird, setzt die Fensterprozedur die globale Variable `bContinueCalc` auf FALSE und schaltet die Menüoption `Abort` aus.

`SecondThread` überprüft diese Variable `bContinueCalc` bei der Berechnung und verläßt die Schleife, wenn die Variable nicht länger ungleich Null ist. Der Thread setzt dann das Semaphor und übergibt die Meldung `WM_CALC_ABORTED` an das Fenster. Bei Erhalt dieser Meldung schaltet das Arbeitsflächenfenster die Option `Start` wieder ein. Das Semaphor ist bereits gesetzt, `SecondThread` kann also mit einer weiteren Berechnung erst weitermachen, wenn wieder `Start` gewählt wird. Dies ist ein Beispiel für die einfache Kommunikation zwischen Threads.

Beachten Sie, daß das Semaphor nur zum Blockieren und Freigeben des Threads ohne Meldungsqueue verwendet wird. Ein Thread mit Meldungsqueue sollte nicht auf ein Semaphor warten müssen, da dies die 1/10-Sekunden-Regel verletzen könnte. Wenn es absolut notwendig ist, kann ein Thread ohne Meldungsqueue einen mit Meldungsqueue für ganz kurze Zeit mit der Funktion `DosSuspendThread` oder `DosEnterCritSec` anhalten. Dies ist gelegentlich hilfreich, wenn beide Threads auf globale Variablen zugreifen. (In BIGJOB4 ist das beim Zugriff der beiden Threads auf `bContinueCalc` nicht notwendig, weil darauf mit einem Maschinenbefehl zugegriffen wird.)

Der Thread mit Meldungsqueue in BIGJOB4 ruft DosSuspendThread auf, um SecondThread nach dem Verlassen der Meldungsschleife in main und vor dem Löschen des Fensters und der Meldungsqueue anzuhalten.

In Threads denken

Threads ohne Meldungsqueue sind für Presentation Manager-Programme, die Eingaben anders als über die Tastatur oder die Maus erhalten, nahezu unentbehrlich. Ein klares Beispiel dafür ist ein Kommunikationsprogramm. Ein solches Programm würde im Thread mit Meldungsqueue ein Arbeitsfenster haben, daß die Tastaturmeldungen bearbeitet, Zeichen mit DosWrite an die serielle Schnittstelle schickt und (wenn lokales Echo eingeschaltet ist) das Zeichen auch auf die Fensteroberfläche schreibt.

Der Thread ohne Meldungsqueue liest die Schnittstelle mit der Funktion DosRead. Wenn sie sehr effektiv eingesetzt wird, kehrt diese Funktion nicht eher zurück, als bis ein Zeichen von der seriellen Schnittstelle gelesen wurde. Ein Thread mit Meldungsqueue sollte DosRead nicht für die Eingabe von einer seriellen Schnittstelle verwenden, weil dadurch die 1/10-Sekunden-Regel verletzt werden könnte. Wenn der Thread ohne Meldungsqueue ein Zeichen liest, kann er eine benutzerdefinierte Meldung an das Fenster übergeben. Das Arbeitsfenster verarbeitet die Meldung durch Anzeige des Zeichens im Fenster.

Ein Presentation Manager-Programm, das Queues für die Kommunikation zwischen Prozessen verwendet, sollte auch zum Lesen einer Queue einen Thread ohne Meldungsqueue anlegen. Der Thread ohne Meldungsqueue ruft die Funktion DosReadQueue mit auf Null gesetztem No-Wait-Flag auf, so daß der Thread blockiert wird, bis etwas in der Queue ist.

In BIGJOB4 wird der zweite Thread immer dann angelegt und freigegeben, wenn er die große Aufgabe erledigen muß. Wenn ein Programm viele große Aufgaben in vielen verschiedenen Threadfunktionen zu erledigen hat, würde es am besten sein, nicht alle diese Threads schon zu Beginn anzulegen, sondern jeden einzelnen erst dann, wenn er benötigt wird. Wenn ein Thread seine große Aufgabe erledigt hat, kann er eine Meldung an das Arbeitsfenster übergeben und sich selbst durch einen Aufruf von DosExit mit 0 als erstem Parameter beenden.

Objektfenster

BIGJOB4 zeigt, wie ein Programm große Aufgaben mit einem Thread ohne Meldungsqueue erledigen kann. Ein Presentation Manager-Programm kann große Aufgaben auch in einem zweiten Thread mit Meldungsqueue erledigen, der »Objektfenster« anlegt.

Objektfenster werden in einem Thread mit Meldungsqueue genauso erzeugt wie normale Fenster. Sie haben eine Fensterprozedur, die die Meldungen an das Objektfenster bearbeitet, wiederum genauso wie bei normalen Fenstern. Objektfenster erscheinen jedoch nicht auf dem Bildschirm und erhalten keine Tastatur- oder Mauseingaben. Tatsächlich erhalten Objektfenster im Prinzip nur zwei Meldungen: WM_CREATE und WM_DESTROY. Die Meldung WM_CREATE wird beim Aufruf von WinCreateWindow geschickt, der das Objektfenster erzeugt; WM_DESTROY wird bei dem Aufruf von WinDestroyWindow geschickt, der das Fenster wieder entfernt. (In der Vorabversion des Presentation Managers, die ich für diesen Artikel verwendet habe, erhält das Objektfenster parallel zum Aufruf von WinCreateWindow auch die Meldung WM_ADJUSTWINDOWPOS.)

Normalerweise verwendet man Objektfenster besonders für den Empfang von Meldungen von anderen Fenstern, vielleicht, um eine objektorientierte Programmiersprache zu implementieren. Doch Objektfenster können auch bei der Bearbeitung großer Aufgaben hilfreich sein. Wenn ein Thread nur Objektfenster erzeugt, gibt es nur eine sehr beschränkte Anzahl von Meldungen, die die Fensterprozedur erhalten kann. Dies vereinfacht die Bearbeitung dieser Meldungen in der Fensterprozedur des Objektfensters ganz erheblich.

Beim Einsatz von Objektfenstern für große Aufgaben ergeben sich mehrere Vorteile. Das Objektfenster ist nicht auf eine Untermenge der Presentation Manager-Funktionen beschränkt. Sie können also für die Kommunikation zwischen dem Arbeitsfenster und dem Objektfenster Meldungen verwenden. Wenn Sie glauben, daß eine meldungsorientierte Architektur der traditionellen Top-Down-Architektur durchweg überlegen ist, werden sie dies vorziehen. Die Verwendung von Objektfenstern in getrennten Threads paßt besser zur Architektur des Presentation Managers als einfache Threads ohne Meldungsqueue.

Die Verwendung eines Objektfensters in einem zweiten Thread ohne Meldungsqueue wird vom Programm BIGJOB5 im *Listing 6* demonstriert. In der Datei BIGJOB5.C werden sechs Meldungen definiert, die das Arbeitsfenster und das Objektfenster für die gegenseitige Kommunikation verwenden.

BIGJOB5 legt den zweiten Thread generell nach der Erzeugung des Hauptfensters des Programms an. Die Funktion SecondThread von BIGJOB5 sieht wie die main-Funktion aus. Es ruft allerdings nicht die Funktion WinInitialize auf, es registriert jedoch eine Fensterklasse und erzeugt ein Fenster. Der Aufruf von WinCreateWindow zur Erzeugung eines Objektfensters ist sehr einfach: Der Parameter übergeordnetes Fenster (parent window) wird auf HWND_OBJECT gesetzt, als zweiter Parameter wird der Name der Fensterklasse verwendet und alle anderen Parameter werden auf 0 oder NULL gesetzt.

Wenn Sie aus dem Menü von BIGJOB5 Start auswählen, übergibt das Arbeitsfenster die Meldung WM_START_CALC an das Objektfenster. Auf ähnliche Art wird bei der Auswahl von Abort aus dem Menü von BIG-

JOB5 vom Arbeitsfenster die Meldung WM_ABORT_CALC übergeben. Beachten Sie auch, daß ClientWndProc auch die normale WM_CLOSE-Meldung bearbeitet (die durch Auswahl der Option Close aus dem Systemmenü erzeugt wird), indem es die Meldung WM_QUIT an das Objektfenster übergibt.

Die Funktion ObjectWndProc beginnt mit der großen Aufgabe als Reaktion auf die Meldung WM_START_CALC. Bei der Durchführung der Berechnung kann sie nur mit einer von zwei Meldungen unterbrochen werden, die in ihrer Meldungsqueue hinterlegt werden. Dabei handelt es sich um WM_ABORT_CALC und WM_QUIT. In beiden Fällen kann die Berechnung bedingungslos abgebrochen werden.

In der Berechnungsschleife ruft ObjectWndProc die Funktion WinQueryQueueStatus auf und prüft das Bit QS_POSTMSG des Rückgabewerts. (Es sollte eigentlich nur nötig sein, zu prüfen, ob der Rückgabewert ungleich Null ist, doch in der von mir verwendeten Vorabversion des Presentation Managers war diese Funktion noch ein wenig fehlerhaft.) Diese Überprüfung von WinQueryQueueStatus ist fast genauso einfach wie die Überprüfung von bContinueCalc in BIGJOB4, doch nicht annähernd so kompliziert wie der Programmteil, der bei Verwendung von WinPeekMsg in BIGJOB3 notwendig war.

ObjectWndProc informiert ClientWndProc darüber, daß es fertig ist oder abgebrochen wurde, indem es die Meldungen WM_CALC_DONE bzw. WM_CALC_ABORTED übergibt. ClientWndProc verwendet diese Meldungen dazu, sein Menü für weitere Berechnungen einzustellen.

Die Beendigung des Programms auf saubere Art und Weise (daß heißt, sowohl der Hauptthread als auch SecondThread entfernen ihre eigenen Fenster und Meldungsqueues) ist knifflig. ClientWndProc bearbeitet die Meldung WM_CLOSE, indem es die Meldung WM_QUIT an das Objektfenster übergibt. Das veranlaßt ObjectWndProc zum Abbruch der großen Aufgabe. Wenn diese WM_QUIT-Meldung aus der Meldungsqueue von SecondThread gelesen wird, gibt WinGetMsg 0 zurück. SecondThread entfernt sein Fenster sowie seine Meldungsqueue und übergibt die Meldung WM_OBJECT_DESTROYED an ClientWndProc. Bei Erhalt dieser Meldung übergibt ClientWndProc die Meldung WM_QUIT an sich selbst und beendet das Programm normal.

Der Einsatz eines Objektfensters für große Aufgaben ist sicherlich komplexer als die Verwendung eines Threads ohne Meldungsqueue, es kann also durchaus sein, daß sie das nicht mögen. Darüber hinaus ist der Objektfenster-Ansatz nicht für alle großen Aufgaben gleich gut geeignet. Wenn ein zweiter Thread zum Beispiel DosRead verwenden muß, um Eingaben von einer Schnittstelle zu lesen oder DosReadQueue, um Eingaben aus einer Queue zu lesen, dann wird der Thread in der OS/2-Funktion blockiert und kann dann natürlich den Status der Meldungsqueue nicht mit WinQueryQueueStatus überprüfen.

BIGJOB5: Make-Datei

```
bigjob5.obj : bigjob5.c bigjob.h
     cl -c -FPa -G2sw -W3 -Zp bigjob5.c

bigjob.res : bigjob.rc bigjob.h
     rc -r bigjob.rc

bigjob5.exe : bigjob5.obj bigjob5.def bigjob.res
     link bigjob5, /align:16, /map,
          /nod slibc slibcp slibfa os2, bigjob5
     rc bigjob.res bigjob5.exe
```

BIGJOB5.DEF: Modul-Definitionsdatei

```
NAME           BIGJOB5
DESCRIPTION    'BigJob Demo Program No. 5 (C) C. Petzold, 1988'
HEAPSIZE       1024
STACKSIZE      8192
EXPORTS        ClientWndProc
               ObjectWndProc
```

BIGJOB5.C: Ein Objektfenster für die große Aufgabe

```
#define INCL_WIN
#define INCL_DOS

#include <os2.h>
#include <math.h>
#include <stdio.h>
#include "bigjob.h"

#define WM_OBJECT_CREATED     (WM_USER + 0)
#define WM_START_CALC         (WM_USER + 1)
#define WM_ABORT_CALC         (WM_USER + 2)
#define WM_CALC_DONE          (WM_USER + 3)
#define WM_CALC_ABORTED       (WM_USER + 4)
#define WM_OBJECT_DESTROYED   (WM_USER + 5)

VOID  FAR SecondThread (VOID) ;
ULONG EXPENTRY ObjectWndProc (HWND, USHORT, ULONG, ULONG) ;

HWND  hwndClient, hwndObject ;
UCHAR cThreadStack [8192] ;


INT main (VOID)
     {
     static CHAR szClassName [] = "BigJob5" ;
     HMQ         hmq ;
     HWND        hwndFrame ;
     QMSG        qmsg ;
     TID         idThread ;

     hab = WinInitialize (0) ;
     hmq = WinCreateMsgQueue (hab, 0) ;

     WinRegisterClass (hab, szClassName, ClientWndProc,
                              CS_SIZEREDRAW, 0, NULL) ;

     hwndFrame = WinCreateStdWindow (HWND_DESKTOP,
                    WS_VISIBLE | FS_SIZEBORDER | FS_TITLEBAR
                    | FS_SYSMENU   | FS_MINMAX   | FS_MENU,
                    szClassName, "BigJob Demo No. 5",
                    0L, NULL, ID_RESOURCE, &hwndClient) ;

     EnableMenuItem (hwndClient, IDM_START, FALSE) ;

     if (DosCreateThread (SecondThread, &idThread,
                          cThreadStack + sizeof cThreadStack))
```

Listing 6

```
          WinAlarm (HWND_DESKTOP, WA_ERROR) ;

     while (WinGetMsg (hab, &qmsg, NULL, 0, 0))
          WinDispatchMsg (hab, &qmsg) ;

     WinDestroyWindow (hwndFrame) ;
     WinDestroyMsgQueue (hmq) ;
     WinTerminate (hab) ;

     return 0 ;
     }


ULONG EXPENTRY ClientWndProc (HWND hwnd, USHORT msg, ULONG mp1,
                                                      ULONG mp2)
     {
     static SHORT iCalcRep, iCurrentRep = IDM_10 ;
     static SHORT iStatus = STATUS_READY ;
     static ULONG lElapsedTime ;

     switch (msg)
          {
          case WM_OBJECT_CREATED:

               EnableMenuItem (hwnd, IDM_START, TRUE) ;
               break ;

          case WM_COMMAND:

               switch (LOUSHORT (mp1))
                    {
                    case IDM_10:
                    case IDM_100:
                    case IDM_1000:
                    case IDM_10000:
                         CheckMenuItem(hwnd,iCurrentRep,FALSE);
                         iCurrentRep = LOUSHORT (mp1) ;
                         CheckMenuItem(hwnd,iCurrentRep,TRUE);
                         break ;

                    case IDM_START:
                         EnableMenuItem(hwnd,IDM_START,FALSE);
                         EnableMenuItem(hwnd,IDM_ABORT,TRUE);

                         iStatus = STATUS_WORKING ;
                         WinInvalidateRect (hwnd, NULL) ;

                         iCalcRep = iCurrentRep ;
                         WinPostMsg (hwndObject, WM_START_CALC,
                                   MAKEULONG(iCalcRep, 0), 0L);
                         break ;

                    case IDM_ABORT:
                         WinPostMsg (hwndObject, WM_ABORT_CALC,
                                                      0L, 0L) ;

                         EnableMenuItem(hwnd,IDM_ABORT,FALSE);
                         break ;

                    default:
                         break ;
                    }
               break ;

          case WM_CALC_DONE:

               iStatus = STATUS_DONE ;
               lElapsedTime = mp1 ;
               WinInvalidateRect (hwnd, NULL) ;

               EnableMenuItem (hwnd, IDM_START, TRUE) ;
               EnableMenuItem (hwnd, IDM_ABORT, FALSE) ;
               break ;

          case WM_CALC_ABORTED:

               iStatus = STATUS_READY ;
               WinInvalidateRect (hwnd, NULL) ;
```

Listing 6 *(Fortsetzung)*

```
               EnableMenuItem (hwnd, IDM_START, TRUE) ;
               break ;

          case WM_PAINT:

               PaintWindow(hwnd,iStatus,iCalcRep,lElapsedTime);
               break ;

          case WM_CLOSE:

               if (hwndObject)
                    WinPostMsg (hwndObject, WM_QUIT, 0L, 0L) ;
               else
                    WinPostMsg (hwnd, WM_QUIT, 0L, 0L) ;
               break ;

          case WM_OBJECT_DESTROYED:

               WinPostMsg (hwnd, WM_QUIT, 0L, 0L) ;
               break ;

          default:
               return WinDefWindowProc (hwnd, msg, mp1, mp2) ;
          }
     return 0L ;
     }

VOID FAR SecondThread ()
     {
     static CHAR szClassName [] = "BigJob5.Object" ;
     HMQ         hmq ;
     QMSG        qmsg ;

     hmq = WinCreateMsgQueue (hab, 0) ;

     WinRegisterClass(hab,szClassName,ObjectWndProc,0L,0,NULL);

     hwndObject = WinCreateWindow (HWND_OBJECT, szClassName,
               NULL, 0L, 0, 0, 0, 0, NULL, NULL, 0, NULL, NULL) ;

     WinPostMsg (hwndClient, WM_OBJECT_CREATED, 0L, 0L) ;

     while (WinGetMsg (hab, &qmsg, NULL, 0, 0))
          WinDispatchMsg (hab, &qmsg) ;

     WinDestroyWindow (hwndObject) ;
     WinDestroyMsgQueue (hmq) ;

     WinPostMsg (hwndClient, WM_OBJECT_DESTROYED, 0L, 0L) ;

     DosExit (0, 0) ;
     }


ULONG EXPENTRY ObjectWndProc (HWND hwnd, USHORT msg, ULONG mp1,
                                                     ULONG mp2)
     {
     double A ;
     SHORT  i, iCalcRep ;
     LONG   lQueueStatus, lTime ;

     switch (msg)
          {
          case WM_START_CALC:

               iCalcRep = LOUSHORT (mp1) ;
               lTime = WinGetCurrentTime (hab) ;

               for (A = 1.0, i = 0 ; i < iCalcRep ; i++)
                    {
                    lQueueStatus =
                         WinQueryQueueStatus (HWND_DESKTOP) ;

                    if (lQueueStatus & QS_POSTMSG)
                         break ;

                    A = Savage (A) ;
                    }
```

Listing 6 *(Fortsetzung)*

```
          if (lQueueStatus & QS_POSTMSG)
               break ;

          lTime = WinGetCurrentTime (hab) - lTime ;

          WinPostMsg(hwndClient, WM_CALC_DONE, lTime, ØL);
          break ;

     case WM_ABORT_CALC:

          WinPostMsg(hwndClient, WM_CALC_ABORTED, ØL, ØL);
          break ;

     default:
          return WinDefWindowProc (hwnd, msg, mp1, mp2) ;
     }
return ØL ;
}
```

Listing 6 *(Ende)*

Nie mehr »Bitte warten«

Wir begannen damit, daß wir uns das Programm BIGJOB1 angesehen haben, daß die ihm übertragene Aufgabe zwar erledigte, doch es auf eine Art und Weise tat, die für den Benutzer in keiner Weise vorteilhaft war. Unsere sofortige Ablehnung dieses Programms und unsere Suche nach besseren Wegen zur Erledigung von großen Aufgaben zeigt, daß wir unsere Einschätzung von einem guten Verhalten eines Anwendungsprogramms geändert haben.

In einer traditionellen Singletasking-Umgebung ohne Fenster, *muß man natürlich* darauf warten, bis ein Datenbankprogramm mit der Sortierung einer Datei fertig ist. Wenn man die Sortierung startet, ist es Zeit für eine Kaffeepause.

In einer traditionellen Multitasking-Umgebung kann es sein, daß das Datenbankprogramm im Hintergrund sortieren und man selbst mit einem anderen Programm arbeiten kann, während man darauf wartet, daß die Sortierung beendet wird.

In einer Multitasking-Fensterumgebung wie dem Presentation Manager sind wir jedoch nur dann zufrieden, wenn der Benutzer weiter mit dem System arbeiten kann, selbst wenn es große Aufgaben durchführt. Ganz klar erfordert die Komplexität der derartigen Strukturierung eines Programms etwas zusätzliche Arbeit vom Programmierer. Doch dadurch wird das Programmm besser einsetz- und benutzbar.

Genauso wie wir nicht länger Programme tolerieren können, die vom Benutzer das Erlernen unzähliger Befehle erfordern, können wir es nicht länger tolerieren, daß Programme einfach die Meldung »Bitte warten« anzeigen und dann den Benutzer warten lassen, bis die große Aufgabe erledigt ist.

Charles Petzold

Impressum

Das *Microsoft System Journal* erscheint alle zwei Monate (ungerade Monatszahlen) etwa Mitte des Vormonats.

Herausgeber, verantwortlich und Anschrift der Redaktion:
Microsoft GmbH, Redaktion *Microsoft System Journal*,
Erdinger Landstr. 2, D-8011 Aschheim-Dornach
Telefon: 089 / 46107-0, Teletex/Telex: (17) 89 83 28, Telefax: 90 63 55

Redaktion:
Günter Jürgensmeier, Haar und Hartmut Niemeier, Wildenberg

Mitarbeiter dieser Ausgabe:
Said Baloui, Michael Bülow, Greg Comeau, Günter Jürgensmeier, Hartmut Niemeier, Charles Petzold, Michael Tischer, Paul Yao

Manuskripteinsendungen:
Manuskripte und Programmlistings werden von der Redaktion gerne angenommen. Mit der Einsendung von Manuskripten und Listings gibt der Verfasser die Zustimmung zum Abdruck und zur Vervielfältigung der Programmlistings auf Datenträgern. Honorare nach Vereinbarung. Für unverlangt eingesandte Manuskripte und Listings wird keine Haftung übernommen. Nicht zur Veröffentlichung gelangte Manuskripte und Listings können nur zurückgeschickt werden, wenn Rückporto beiliegt.

Titelgestaltung: Hermann Menig
Anzeigenverkauf: Marianne Nuß
Druck und Abonnements:
schury praxisformulare GmbH, Postfach 270, D-8200 Rosenheim
Bezugspreise: Das Einzelheft kostet DM 19,80. Der Abonnementpreis beträgt DM 115,- für 6 Ausgaben und DM 210,- für 12 Ausgaben. Zu den einzelnen Ausgaben ist zum Preis von DM 19,80 eine Diskette mit allen Listings erhältlich. Das Abonnement inklusive Diskette kostet DM 230,- für 6 Ausgaben und DM 420,- für 12 Ausgaben. In den Preisen enthalten sind Mehrwertsteuer, Versandkosten und Zustellgebühren. Auslandsbezug auf Anfrage. Sollte die Zeitschrift aus Gründen, die nicht vom Herausgeber zu vertreten sind, nicht geliefert werden können, besteht kein Anspruch auf Nachlieferung oder Erstattung vorausbezahlter Bezugsgelder.
Bezugsmöglichkeiten: In Buchhandlungen und im Computer-Fachhandel. Abonnements und Einzelbestellungen: schury GmbH.

Das *Microsoft System Journal* wird mit Microsoft Word 4.0 geschrieben und gestaltet. Der Ausdruck erfolgt mit den HP-Softfonts AD und AF und dem Programm- und Schriftenpaket *DocuJet* auf einem HP-LaserJet Series II.

Eine komfortable Benutzeroberfläche in C (Teil 1):

Bildschirm- und Fensterverwaltung in C

In einer Serie möchten wir Ihnen ab dieser Ausgabe mehrere C-Module vorstellen, die Ihnen die Möglichkeit bieten, Ihre C-Programme mit einer SAA- bzw. DOS 4.0-konsistenten Benutzeroberfläche zu versehen und ein Programm dadurch optimal an diese neue DOS-Version anzupassen.

Mittelfristig wird der überwiegende Teil der DOS-Anwender auf die neue Version 4.0 umsteigen und ihre integrierte Benutzeroberfläche zu bedienen und schätzen lernen. Mit dieser Oberfläche lernt der Anwender eine Bedienerführung kennen, die sich fortschrittlicher Interaktionsmechanismen wie Fenster, Pull-down-Menüs und File-select-Boxen bedient. Da liegt die Idee nahe, diese Oberfläche auch auf eigene Programme zu übertragen und dadurch nicht nur den Bedienungskomfort zu steigern, sondern gleichzeitig auch die Einarbeitungszeit des Anwenders zu minimieren.

Konzeption der Serie

Dieser Idee folgend wird in dieser Folge eine Bildschirm- und Fensterverwaltung vorgestellt, der im zweiten Teil der Serie ein Modul zur Maus- und Tastaturverwaltung folgt. In der dritten Folge wird auf der Basis dieser beiden Module eine Pull-down-Menüverwaltung erarbeitet, die der von DOS 4.0 nachgebildet ist und neben der Tastatur auch die Maus als vollwertiges Eingabegerät unterstützt.

Alle Module werden unter dem Microsoft C-Compiler, Version 5.1, entwickelt und sind in Verbindung mit allen Speichermodellen dieses Compilers voll einsatzfähig. In jeder Folge finden Sie neben dem Source-Listing des eigentlichen Moduls und der zugehörigen Include-Datei auch ein Demo-Programm, das die Arbeit mit den einzelnen Funktionen des Moduls verdeutlichen und Ihnen den Weg zur Einbindung dieser Funktionen in eigene Programme weisen soll.

Auch wenn Sie der Idee, eine DOS 4.0-konsistente Benutzeroberfläche in ihren Programmen nachzubilden, nicht folgen möchten, werden Sie feststellen, daß die hier vorgestellten Funktionen in vielen Ihrer Programme sinnvoll eingesetzt werden können. Neben der Steigerung des Bedienungskomforts bringt der Einsatz vorgefertigter Module auch eine nicht unwesentliche Reduzierung des Entwicklungsaufwandes mit sich, so daß sich das Abtippen (bzw. Bestellen der Begleit-Diskette) gleich in doppelter Hinsicht lohnt.

Die Bildschirm- und Fensterverwaltung

Die Terminal-orientierte Bildschirmausgabe über `printf()` ist schon lange »out«, die Bildschirmausgabe über das BIOS viel zu langsam und die Entwicklung entsprechender Assemblerroutinen nicht jedermanns Sache.

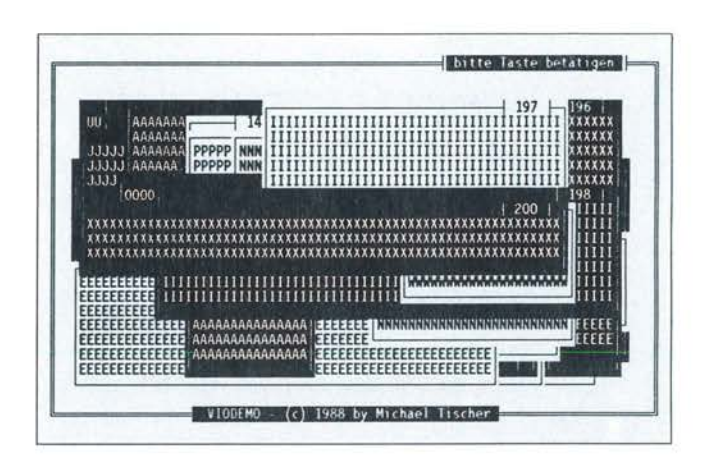


Bild 1: *Das VIO-Modul in Aktion.*

Wie also Abhilfe schaffen, wenn auch ein noch so gutes Programm dem Konkurrenzdruck auf dem PC-Markt nur noch standhalten kann, wenn es über ein ansprechendes und schnelles Bildschirminterface verfügt, das sich am Stand der Technik - eben der »Fenstertechnik« - orientiert? Wer sich mit der Funktionsweise der Video-Karten des PCs ein wenig auskennt, dem fällt die Antwort nicht schwer: durch direkten Zugriff auf den Video-RAM!

Alle Videokarten, die im PC-Bereich zum Einsatz kommen, von MDA und CGA bis hin zu VGA, verfügen über einen als »Video-RAM« bezeichneten RAM-Bereich, der im Textmodus die ASCII-Codes der Zeichen auf den einzelnen Bildschirmpositionen und ihre Farbe bzw. Attribut aufnimmt. Da sich dieser RAM-Bereich, je nach der Art Videokarte, an der Segmentadresse 0xB000 oder 0xB800 in den normalen PC-RAM-Speicher eingliedert, kann auf ihn wie auf jeden anderen Speicherbereich zugegriffen werden. Ein Problem entsteht dabei nur im Fall der IBM-CGA-Karte, bei der der Zugriff auf den Video-RAM mit dem Bildschirmaufbau durch den Video-Controller synchronisiert werden muß. Unterbleibt diese Synchronisation entsteht eine Art »Krispeln«, das sich durch kleine weiße Streifen auf dem Bildschirm bemerkbar macht.

Da sich diese Karte heute aber kaum noch im Einsatz befindet und die CGA-Karten anderer Hersteller dieses Problem nicht aufweisen, kann es bei den folgenden Betrachtungen vernachlässigt werden.

Entscheidend für den Zugriff auf den Video-RAM ist die Kenntnis um seinen Aufbau. Hier ist zu beachten, daß jede Bildschirmposition innerhalb des Video-RAM zwei aufeinanderfolgende Speicherstellen bzw. Bytes in Anspruch nimmt. Das erste Byte nimmt dabei den ASCII-Code des Zeichens und das zweite Byte sein Attribut bzw. seine Farbe auf. Die Größe des Video-RAM ergibt sich dadurch aus der Anzahl der auf dem Bildschirm dargestellten Zeichen multipliziert mit 2. Im 80*25-Zeichen-Modus mit seinen 25 Bildschirmzeilen sind es 4000 Bytes, im 80*43-Zeichen-Modus der EGA-Karte sind es 6880 Bytes und im 80*50-Zeichen-Modus der VGA-Karte schließlich 8000.

Ungeachtet der Anzahl der auf dem Bildschirm dargestellten Zeilen, nimmt das Zeichen in der oberen linken Bildschirmecke immer die erste Position im Video-RAM, also die Bytes mit der Offsetadresse 0x0000 und 0x0001, ein. Darauf folgt das Zeichen in der zweiten Bildschirmspalte der ersten Zeile, das die Bytes mit den Offsetadressen 0x0002 und 0x0003 belegt. An dieses Zeichen schließen sich die übrigen 78 Zeichen der ersten Zeile an, so daß diese Zeile die ersten 160 Bytes im Video-RAM in Anspruch nimmt.

Unmittelbar auf die erste Zeile folgt im Video-RAM die zweite Zeile, die, wie auch alle darauffolgenden Zeilen, ebenfalls 160 Bytes belegt. Durch diese Anordnung der Zeichen und ihrer Attribute innerhalb des Video-RAM läßt sich die Offsetposition eines Zeichens leicht mit Hilfe folgender Formel bestimmen:

Offset = Spalte * 2 + Zeile * 160

An dieser Offsetposition findet man den ASCII-Code des Zeichens und an der darauffolgenden Position das zugehörige Attribut- bzw. Farb-Byte. Zwar wird in dieser Formel von einem 80-Spalten-Bildschirm ausgegangen, doch kann man sie leicht an jede beliebige Bildschirmauflösung anpassen, indem man die Konstanten 160 durch den Ausdruck

(Anzahl der Spalten pro Zeile * 2)

ersetzt.

Das Verständnis dieses Aufbaus allein reicht aus, um den theoretischen Hintergrund der einzelnen Funktionen aus dem hier vorgestellten Modul zu verstehen.

Das C-Modul VIO.C

Alle Funktionen des Moduls zur Bildschirm- und Fensterverwaltung befinden sich innerhalb der Datei VIO.C (*Listing 2*). Um die Funktionen dieses Moduls innerhalb Ihrer Programme aufrufen zu können, müssen zwei Schritte vollzogen werden: Ähnlich wie bei den Funktionen aus der Bibliothek Ihres Microsoft C-Compilers, sollten Sie zunächst die Datei VIO.H (*Listing 1*) über den `#include`-Befehl in Ihr C-Programm einbinden. In ihr befinden sich die Deklarationen der einzelnen Funktionen aus dem VIO-Modul sowie eine ganze Reihe von Typ-, Konstanten- und Makrodefinitionen, die Sie zum Aufruf der einzelnen Funktionen benötigen.

Beim Aufruf des Compilers zur Kompilierung Ihres Programms müssen Sie dann neben dem Namen Ihres Programms auch den Namen des VIO-Moduls angeben, damit dieses ebenfalls kompiliert und während des Link-Vorgangs mit Ihrem Programm verbunden wird.

Eine zentrale Rolle kommt innerhalb dieses Moduls den Strukturen `windes`, `velb` und `velw` zu, wobei letztere in der Include-Datei VIO.H definiert und dort zu der Union `vel` zusammengeschlossen werden. `Vel` steht dabei für »Video-ELement« und beschreibt die Darstellung einer

```
/*
[ ]---------------------------------------------------------[ ]
 |                                                           |
 |      Include-Datei     : VIO.H                            |
 |      zur Einbindung    : der Funktionen zur Bildschirm-   |
 |                          verwaltung                       |
 |      erstellt am       : 23.11.1988                       |
 |      letztes Update am:  6.12.1988                        |
 |                                                           |
 |      (Copyright)       : 1988 by MICHAEL TISCHER          |
 |                                                           |
[ ]---------------------------------------------------------[ ]
*/
/*-- Typedefs ----------------------------------------------*/
typedef unsigned char BYTE;      /* wir basteln uns ein BYTE */
typedef unsigned int WORD;                   /* dito: WORD */
typedef union vel VEL;                /* nur als Abkürzung */

/*-- Strukturen --------------------------------------------*/
struct velb {          /* eine Bildschirmposition als 2 Bytes */
             BYTE zeichen,                 /* der ASCII-Code */
                  attribut;      /* das zugehörige Attribut */
            };

struct velw {           /* eine Bildschirmposition als 1 Word */
             WORD inhalt;    /*  ASCII-Zeichen und Attribut */
            };

union vel {             /* beschreibt eine Bildschirmposition */
           struct velb h;
           struct velw x;
          };

/*-- Konstanten --------------------------------------------*/
#define TRUE 1
#define FALSE 0

#define SCHWARZ    0   /* Farben für Color-Karten ----------*/
#define BLAU       1
#define GRUEN      2
#define CYAN       3
#define ROT        4
#define VIOLETT    5
#define BRAUN      6
#define DGRAU      7
#define HGRAU      8
#define HBLAU      9
#define HGRUEN    10
#define HCYAN     11
#define HROT      12
#define HVIOLETT  13
#define GELB      14
#define WEISS     15
#define BLINKEN  128

#define NORMAL     0x07   /* Attribute für Mono-Karten -----*/
#define HNORMAL    0x0f         /* besonders helle Schrift */
#define INVERS     0x70                /* schwarz auf weiß */
#define UNDERLINE  0x01 /* unterstrichenes normales Zeichen */
#define UNVISI     0x00             /* schwarz auf schwarz */
#define IUNVISI    0x77                   /* weiß auf weiß */

#define EINRA      0          /* Rahmentypen für VioFrame */
#define DOPRA      1
#define VOLLRA     2
#define PUNKTRA    3
#define KEINRA  0xff                /* keinen Rahmen ziehen */

#define NOCLEAR 0xff          /* kein Löschen der Leerzeilen */
                              /* bei den Scroll-Funktionen   */

/*-- Codes für die verschiedenen Videokarten ---------------*/
#define MDA        0                       /* MDA und HGC */
#define CGA        1
#define EGA        2
#define EGA_MONO   3                /* EGA an MDA-Monitor */
#define VGA        4
#define VGA_MONO   5       /* VGA an analogem Mono-Monitor */
#define MCGA       6
#define MCGA_MONO  7      /* MCGA an analogem Mono-Monitor */
```

Listing 1: `VIO.H`

```
/*-- Macros ---------------------------------------------------*/

#define COL(v, h) ( (h) << 4 | (v) )  /* definiert Farbbyte */
#define BLINK( c ) ( ( c ) | BLINKEN )
#define BUFLEN(x1, y1, x2, y2) \
        ( ((x2)-(x1)+1) * ((y2)-(y1)+1) * sizeof( VEL ) )
#define AKTZ ( vzeile )                    /* die aktuelle Zeile */
#define AKTS ( vspalte )                  /* die aktuelle Spalte */
#define VL(i) (( i ) + viewx1) /* linke Spalte im View-Ber. */
#define VO(i) (( i ) + viewy1)  /* obere Zeile im View-Ber. */
#define VR(i) (( i ) + viewx2)  /* rechte Spalte im View-B. */
#define VU(i) (( i ) + viewy2) /* untere Zeile im View-Ber. */
#define VCOL ( viewx2 - viewx1 + 1 )  /* Spalten im View-B. */
#define VROW ( viewy2 - viewy1 + 1 )   /* Zeilen im View-B. */


/*-- Funktionsdeklarationen über Makros -------------------*/

#define VioClear(x1,y1,x2,y2,f)   VioFill(x1,y1,x2,y2,' ',f)
#define VioClearScreen( f ) \
        VioFill(Ø ,Ø , anzcol-1, anzline-1, ' ', f)
#define VioScrollLeft(x1, y1, x2, y2, anz, f)\
        VioScrollHori( x1, y1, x2, y2, anz, f, TRUE );
#define VioScrollRight(x1, y1, x2, y2, anz, f)\
        VioScrollHori( x1, y1, x2, y2, anz, f, FALSE );
#define VioGetSystem() ( vkarte )
#define VioIsColor() ( color )
#define VioSetLines(x) ( anzline = (x) )
#define VioGetLines() ( anzline )
#define VioSetCols(x) ( anzcol = (x) )
#define VioGetCols() ( anzcol )
#define VioHideCursor() VioSetCursor( Ø, anzline )

/*-- externe Variablen ------------------------------------*/

extern BYTE viewx1,   /* obere linke Ecke des View-Bereich  */
            viewy1,   /* in Bezug auf den ganzen Bildschirm */
            viewx2, /* untere rechete Ecke des View-Bereich */
            viewy2, /* in Bezug auf den ganzen Bildschirm   */
            vzeile,    /* die aktuelle Cursorposition in    */
            vspalte,   /* Bezug auf den gesamten Bildschirm */
            vkarte,      /* Code für die aktive Video-Karte */
            color,           /* TRUE, wenn Farbdarstellung */
            anzline,        /* Anzahl der Bildschirmzeilen */
            anzcol;        /* Anzahl der Bildschirmspalten */

/*-- Funktions-Deklarationen ------------------------------*/

void VioInit       ( void );
void VioSetView    ( BYTE x1, BYTE y1, BYTE x2, BYTE y2 );
void VioGetView    ( BYTE * x1, BYTE * y1,
                     BYTE * x2, BYTE * y2 );
void VioGet        ( BYTE x1, BYTE y1, BYTE x2,
                     BYTE y2, VEL far * bptr );
void VioPut        ( BYTE x1, BYTE y1, BYTE x2,
                     BYTE y2, VEL far * bptr);
BYTE VioWinOpen    ( BYTE x1, BYTE y1, BYTE x2, BYTE y2 );
void VioWinClose   ( BYTE redraw );
void VioSetCursor  ( BYTE spalte, BYTE zeile );
BYTE VioGetCurCol  ( void );
BYTE VioGetCurRow  ( void );
void VioPrint      ( BYTE spalte, BYTE zeile, BYTE farbe,
                     BYTE cursor, char * string );
void VioPrintf     ( BYTE spalte, BYTE zeile, BYTE farbe,
                     BYTE cursor, char * string, ... );
char *VioStrep     ( char zeichen, BYTE anz );
void VioFill       ( BYTE x1, BYTE y1, BYTE x2,
                     BYTE y2, char zeichen, BYTE farbe);
void VioFrame      ( BYTE x1, BYTE y1, BYTE x2,
                     BYTE y2, BYTE rahmen, BYTE farbe );
void VioScrollUp   ( BYTE x1, BYTE y1, BYTE x2,
                     BYTE y2, BYTE anzahl, BYTE farbe );
void VioScrollDown( BYTE x1, BYTE y1, BYTE x2,
                     BYTE y2, BYTE anzahl, BYTE farbe );
BYTE VioMoveUp     ( BYTE anzahl );
BYTE VioMoveDown   ( BYTE anzahl );
BYTE VioMoveRight  ( BYTE anzahl );
BYTE VioMoveLeft   ( BYTE anzahl );
void VioColor      ( BYTE x1, BYTE y1, BYTE x2,
                     BYTE y2, BYTE farbe );
```

Listing 1: *(Ende)*

Bildschirmposition innerhalb des Video-RAM. Während `velw` die beiden Bytes, die eine Bildschirmposition repräsentieren, unter dem Namen `inhalte` zu einem Wort zusammenfaßt, können der ASCII-Code des Zeichens und sein Attribut-Byte mit Hilfe der Struktur `velb` individuell über die Variablen `zeichen` und `attribut` angesprochen werden.

Indem der Video-RAM als ein Vektor, bestehend aus Elementen vom Typ der Union `vel`, betrachtet wird, kann auf alle Zeichen individuell zugegriffen werden. Wichtig ist dabei allerdings, daß der Zugriff jeweils über Pointer geschieht, da dieser Vektor (der Video-RAM) nicht zu den Daten des C-Programms zählt, sondern fest im Speicher lokalisiert ist. Pointer, die dem Zugriff auf diesen Vektor dienen, müssen grundsätzlich vom Typ `far` sein, da sich der Video-RAM außerhalb des Datensegments des C-Programms befindet. Ein entsprechender Variablentyp wird innerhalb des VIO-Moduls mit Hilfe des `typedef`-Befehls in Form des Variablentyps `vpfar` (Viodeo-Pointer `far`) definiert.

Die Adresse einer Bildschirmposition kann einem Pointer vom Typ `vpfar` mit Hilfe des Makros `vpos` zugewiesen werden. Er bezieht die Segmentadresse des Video-RAM aus der globalen Variablen `vstart`, zu der er die Offsetadresse des Zeichens addiert. Sie erhält er durch das Makro `vofs`, in dem sich der Aufbau des Video-RAM widerspiegelt.

Hat man mit Hilfe des Makros `vpos` einen Pointer auf die entsprechende Bildschirmposition gesetzt, kann auf diese Bildschirmposition über die Komponenten der Union `vel`, die Strukturen `velw` (mit dem Namen `x`) und `velb` (mit dem Namen `h`), und deren Komponenten zugegriffen werden. Die folgende Codesequenz bringt z.B. den Buchstaben A an die Bildschirmspalte 10 der Zeile 2, wobei der Buchstabe mit grüner Farbe auf schwarzem Grund (Farbcode = 2) erscheint.

```
{
 VPFAR vptr;

 vptr = VPOS( 1Ø, 2 );
 vptr->h.zeichen = 'A';
 vptr->h.attribut = 5;
}
```

Etwas beschleunigen kann man den Zugriff auf die Bildschirmposition zusätzlich dadurch, daß man das Zeichen und das Attribut gleichzeitig über die Struktur `velw` als Wort in die entsprechende Speicherstelle innerhalb des Video-RAM lädt:

```
vptr->x.inhalt = 'A' + (5 << 8);
```

Für die Fensterverwaltung innerhalb des Moduls spielt die Struktur `windes` eine große Rolle. Immer, wenn ein

Fenster auf dem Bildschirm geöffnet wird, wird eine solche Struktur für das neue Fenster angelegt und an das Ende des Fenster-Vektors angehängt. Dieser Vektor, der dynamisch auf dem Heap allokiert wird und dessen Adresse in der globalen Variablen `winptr` verzeichnet ist, enthält für jedes der geöffneten Fenster einen Eintrag vom Typ der Struktur `windes`. Indem Fenster nach dem LIFO-Prinzip geöffnet und wieder geschlossen werden, vergrößert und verkleinert sich dieser Vektor dynamisch. Die maximale Anzahl der gleichzeitig geöffneten Fenster wird dabei nur durch die Größe des freien Speicherplatzes auf dem Heap beschränkt, der neben dem Fenster-Vektor auch jeweils einen Puffer aufnehmen muß, in dem der Bildschirminhalt unter dem Fenster gespeichert wird.

Neben der Adresse dieses Puffers finden sich innerhalb der Struktur `windes` auch die Eckkoordinaten des Fensters sowie die aktuelle Cursorposition zum Zeitpunkt, an dem das Fenster geöffnet wurde. Darüberhinaus werden hier auch die Koordinaten des View-Bereichs gespeichert, von dem später noch die Rede sein soll.

Die Funktionen des VIO-Moduls

Nach diesem Einblick in die Art, wie innerhalb des VIO-Moduls auf den Bildschirm zugegriffen und die einzelnen Fenster verwaltet werden, geben die folgenden Seiten einen Überblick über die einzelnen Funktionen, die das VIO-Modul dem Aufrufer bereitstellt. Die Funktionen werden dabei in der Reihenfolge ihres Auftretens innerhalb des Listings vorgestellt. Während hier allerdings nur die grundsätzliche Aufgabe der einzelnen Funktionen vorgestellt werden kann, können Sie die jeweils zu übergebenden Parameter und ihre Reihenfolge der Include-Datei VIO.H bzw. dem VIO-Modul VIO.C entnehmen.

VioInit

Als erste Funktion aus diesem Modul muß immer die Funktion `VioInit` aufgerufen werden, die das Modul initialisiert. Sie stellt zunächst die Art der aktiven Videokarte fest und speichert einen entsprechenden Code, wie er in den Konstanten CGA, MDA etc. innerhalb der Datei VIO.H verzeichnet ist, in der globalen Variablen `vkarte` ab. Innerhalb des Anwendungsprogramms kann er mit Hilfe des Makros `VioGetSys` (ebenfalls VIO.H) abgefragt werden. Auf der Basis dieser Information lädt `VioInit` dann die Segmentadresse des Video-RAM in die globale Variable `vstart`.

Darüber hinaus setzt sie das `color`-Flag, das anzeigt, ob die Videokarte Farben oder nur Schwarz-Weiß-Attribute anzeigen kann. Dieses Flag kann durch das Makro `VioIsColor` abgefragt werden. Es liefert den Wert ungleich 0 zurück, wenn die Karte Farben darstellen kann. Bei einer monochromen Bildschirmkarte wird dem Aufrufer hingegen der Wert 0 zurückgeliefert.

Damit das VIO-Modul auch mit den neuen Bildschirmauflösungen der EGA- und VGA-Karten einwandfrei zusammenarbeiten kann, ermittelt es zusätzlich die Anzahl der Zeilen und Spalten auf dem Bildschirm und trägt sie in die Variablen `anzline` (Anzahl der Zeilen) und `anzcol` (Anzahl der Spalten) ein. Über die Makros `VioGetLines` und `VioGetCols` können Sie den Inhalt dieser Variablen innerhalb Ihrer Anwendungsprogramme jederzeit abfragen. Umgekehrt können Sie den Inhalt dieser Variablen über die Makros `VioSetLines` und `VioSetCols` ändern. Dies sollte immer dann geschehen, wenn Sie die Anzahl der Zeilen und Spalten auf dem Bildschirm verändern, da einwandfreie Arbeit des VIO-Moduls von der genauen Kenntnis der Bildschirmauflösung abhängig ist.

VioSetView

Diese Funktion definiert den aktuellen View-Bereich, bei dem es sich um einen Bildschirmbereich handelt, der vor allem in Verbindung mit Fenstern zum Einsatz kommt. Er erlaubt die Bezugnahme auf die Eckkoordinaten des aktuellen Fensters anstelle absoluter Bildschirmkoordinaten. Dies ist vor allem dann von Vorteil, wenn ein Fenster über den Bildschirm verschoben wird und seine Bildschirmposition damit nicht konstant ist.

Zwar erwarten alle Funktionen innerhalb des VIO-Moduls Koordinaten, die sich auf den gesamten Bildschirm beziehen, doch kann mit Hilfe der Makros `vl`, `vo`, `vr` und `vu` aus der Datei VIO.H auf die Eckkoordinaten des aktuellen View-Bereichs Bezug genommen werden. Zusätzlich kann die Spaltenbreite des jeweils aktuellen View-Bereichs mit Hilfe des Makros `vcol` und die Anzahl der Zeilen innerhalb des View-Bereichs durch das Makro `vrow` ermittelt werden.

Während `VioInit` den gesamten Bildschirm als View-Bereich deklariert, wird beim Öffnen eines Fensters der View-Bereich mit dem Bildschirmbereich gleichgesetzt, über den sich das Fenster erstreckt.

Eine anderer View-Bereich kann jeweils durch den Aufruf der Funktion `VioSetView` definiert werden.

VioGetView

Im Gegensatz zu `VioSetView` ermittelt `VioGetView` die Eckkoordinaten des aktuellen View-Bereichs und übergibt sie dem Aufrufer in den Variablen, deren Adresse er beim Aufruf von `VioGetView` angegeben hat.

VioGet

Der Inhalt eines Bildschirmbereichs (die Zeichen und ihre Attribute) kann mit Hilfe von `VioGet` in einen Puffer geholt werden, dessen Adresse `VioGet` als Far-Pointer übergeben wird. Die Größe des Puffers kann der Aufrufer mit Hilfe des Makros `buflen` ermitteln, das in der Datei VIO.H definiert wird.

```c
/***********************************************************************/
/*                              V I O . C                             */
/*--------------------------------------------------------------------*/
/*    Aufgabe        : Stellt verschiedene Funktionen zum             */
/*                     Zugriff auf den Bildschirm bereit.             */
/*--------------------------------------------------------------------*/
/*    Autor          : MICHAEL TISCHER                                */
/*    entwickelt am  : 23.11.1988                                     */
/*    letztes Update :  6.12.1988                                     */
/*--------------------------------------------------------------------*/
/*    Erstellung     : CL /A[S|M|C|L|H] VIO.C /C                      */
/*                     dann mit einem anderen Modul linken            */
/***********************************************************************/

/*-- Include-Dateien einbinden ------------------------------------*/

#include "vio.h"
#include <dos.h>
#include <stdlib.h>
#include <memory.h>
#include <stdarg.h>

/*-- Kompilierungs-Befehle ---------------------------------------*/

#pragma pack(1)                    /* Strukturen byteweise packen */

/*-- Typedefs ----------------------------------------------------*/

typedef VEL far * VPFAR;     /* FAR-Pointer in den Video-RAM */
typedef struct windes WINDES;        /* ein Fensterbeschreiber */
typedef WINDES * WIPTR;   /* Pointer auf Fensterbeschreiber */

/*-- Makros ------------------------------------------------------*/

#define MK_FP(seg,ofs) ((void far *)\
                      (((unsigned long)(seg) << 16) | (ofs)))
#define VOFS(x,y) ( anzcol * ( y ) + ( x ) )
#define VPOS(x,y) (VPFAR) ( vstart + VOFS( x, y ) )
#define WINDESLEN ( sizeof(struct windes) )
#define VELLEN ( sizeof( VEL ) )

#ifdef M_I86SM            /* wird im SMALL-Modell kompiliert? */
  #define NEARDATA                          /* Ja, NEAR-Daten */
#endif
#ifdef M_I86MM           /* wird im MEDIUM-Modell kompiliert? */
  #define NEARDATA                          /* Ja, NEAR-Daten */
#endif

#ifdef NEARDATA                                /* NEAR-Daten? */
  #define MOVE(s, d, l) movedata( FP_SEG( (void far *) s ), \
                                  FP_OFF( (void far *) s ), \
                                  FP_SEG( (void far *) d ), \
                                  FP_OFF( (void far *) d ), \
                                  l )
#else                                     /* Nein, FAR-Daten */
  #define MOVE(s, d, l) \
          memmove( (void far *) d, (void far *) s, l)
#endif

/*-- Strukturen --------------------------------------------------*/

struct windes {                          /* Fensterbeschreiber */
    BYTE x1, y1,          /* die Eckkoordinaten des Fensters */
         x2, y2,
         viewx1, viewy1,    /* Koordinaten des View-Bereich */
         viewx2, viewy2,
         curs, curz;        /* Cursorkoordinaten vor Öffnen */
         VEL * winmem;        /* Pointer auf Fenster-Puffer */
   };

/*-- globale Variablen -------------------------------------------*/

VPFAR vstart; /* Pointer auf das erste Zeichen im Video-RAM */
BYTE  vzeile,      /* nimmt die aktuelle Cursorposition auf */
      vspalte,
      viewx1,          /* obere linke Ecke des View-Bereich */
      viewy1,          /* in Bezug auf den ganzen Bildschirm */
      viewx2,         /* untere rechte Ecke des View-Bereich */
```

Listing 2: `VIO.C`

```c
      viewy2,         /* in Bezug auf den ganzen Bildschirm  */
      vkarte,             /* Code für die aktive Videokarte */
      color, /* ist TRUE, wenn Video-Karte Farben darstellt */
      anzline = 25,          /* Anzahl der Bildschirmzeilen */
      anzcol = 80,          /* Anzahl der Bildschirmspalten */
      winopen = 0;         /* Anzahl der geöffneten Fenster */
WIPTR winptr;                 /* Ptr auf Fensterbeschreiber */

/***********************************************************************
*  Funktion          : V i o I n i t                                  *
**--------------------------------------------------------------------**
*  Aufgabe           : Leitet die Arbeit mit dem VIO-Modul            *
*                      ein.                                           *
*  Eingabe-Parameter: keine                                           *
*  Return-Wert       : keiner                                         *
*  Info              : Diese Funktion muß als erste Funktion          *
*                      aus diesem Modul aufgerufen werden.            *
***********************************************************************/

void VioInit( void )
{
 static BYTE vmode[] = {
                        MDA, CGA, 0, EGA, EGA_MONO, 0,
                        VGA_MONO,  VGA, 0, MCGA, MCGA_MONO,
                        MCGA
                       };
 static BYTE egamode[] = {
                         EGA, EGA, EGA_MONO,
                        };

 union REGS regs;       /* Prozessorregs für Interruptaufruf */

 vkarte = 0xff;            /* noch keine Video-Karte entdeckt */

 /*-- testen, ob VGA- oder MCGA-Karte installiert ist ------*/
 regs.x.ax = 0x1a00;                /* Funktion 1Ah des Video- */
 int86(0x10, &regs, &regs);         /* BIOS aufrufen           */
 if ( regs.h.al == 0x1a )                  /* VGA oder MCGA? */
  {                                                  /* Ja */
   vkarte = vmode[ regs.h.bl-1 ];     /* Code aus Tab. holen */
   color = !( vkarte==MDA || vkarte==EGA_MONO );
  }
 else                                  /* weder VGA noch MCGA */
  {                                    /* auf EGA-Karte testen */
   regs.h.ah = 0x12;           /* Funktion 12h Unterfunktion */
   regs.h.bl = 0x10;           /* 10h aufrufen               */
   int86(0x10, &regs, &regs);          /* Video-BIOS aufrufen */
   if ( regs.h.bl != 0x10 )              /* EGA installiert? */
    {                                                /* Ja */
     vkarte = egamode[ (regs.h.cl >> 1) % 3 ]; /* hole Code */
     color = ( vkarte!=EGA_MONO );
    }
  }

 /*-- Pointer auf Video-RAM ermitteln ----------------------*/
 regs.h.ah = 15;            /* aktuellen Video-Modus ermitteln */
 int86(0x10, &regs, &regs);   /* BIOS-Video-Interrupt aufr. */
 vstart = (VPFAR) MK_FP((regs.h.al!=7) ? 0xb800 : 0xb000, 0);
 if ( vkarte == 0xff )           /* weder EGA, VGA oder MCGA? */
  vkarte = (color=(regs.h.al!=7)) ? CGA : MDA;           /* Ja */
 else             /* ist EGA, VGA oder MCGA, Zeilen ermitteln */
  anzline = *((BYTE far *) MK_FP( 0x40, 0x84 )) + 1;
 if ( regs.h.al==0 || regs.h.al==2 )
  color = FALSE;     /* in Color-Modus ohne Farbdarstellung */

 anzcol = *((BYTE far *) MK_FP( 0x40, 0x4a ));    /* Spalten */
 regs.h.ah = 5;          /* aktuelle Bildschirmseite auswählen */
 regs.h.al = 0;                           /* Bildschirmseite 0 */
 int86(0x10, &regs, &regs);       /* BIOS-Video-Int. aufrufen */

 regs.h.ah = 3;            /* aktuelle Cursorposition ermitteln */
 regs.h.bh = 0;            /* Zugriff auf die Bildschirmseite 0 */
 int86(0x10, &regs, &regs);       /* BIOS-Video-Int. aufrufen */
 vzeile = regs.h.dh;                   /* Cursorposition merken */
 vspalte = regs.h.dl;
 VioSetView(0, 0, anzcol-1, anzline-1); /* View-B.=ges. Bs. */
 winptr = (WIPTR) malloc( 1 );  /* ein Byte für Fensterbes. */
}
```

Listing 2: *(Fortsetzung)*

```
/************************************************************
*  Funktion          : V i o S e t V i e w                    *
**---------------------------------------------------------**
*  Aufgabe           : Setzt einen neuen View-Bereich         *
*  Eingabe-Parameter: X1, Y1 = obere linke Ecke des Bereichs *
*                      X2, Y2 = untere rechte Ecke des Ber.   *
*  Return-Wert       : keiner                                 *
************************************************************/

void VioSetView( BYTE x1, BYTE y1, BYTE x2, BYTE y2 )
{
 viewx1 = x1;                 /* Koordinaten in den globalen */
 viewy1 = y1;                 /* View-Variablen merken       */
 viewx2 = x2;
 viewy2 = y2;
}

/************************************************************
*  Funktion          : V i o G e t V i e w                    *
**---------------------------------------------------------**
*  Aufgabe           : Holt die Koordinaten des aktuellen     *
*                      View-Bereichs.                         *
*  Eingabe-Parameter: X1, Y1 = Pointer auf Variablen, die    *
*                      X2, Y2   die Koordinaten des View-     *
*                               Bereichs aufnehmen.           *
*  Return-Wert       : keiner                                 *
************************************************************/

void VioGetView( BYTE * x1, BYTE * y1, BYTE * x2, BYTE * y2 )
{
 *x1 = viewx1;                /* Koordinaten aus den globalen */
 *y1 = viewy1;                /* View-Variablen holen         */
 *x2 = viewx2;
 *y2 = viewy2;
}

/************************************************************
*  Funktion          : V i o G e t                            *
**---------------------------------------------------------**
*  Aufgabe           : Holt einen Bildschirmbereich in einen  *
*                      Puffer.                                *
*  Eingabe-Parameter: X1, Y1 = obere linke Ecke des Bereichs *
*                      X2, Y2 = untere rechte Ecke des Ber.   *
*                      BPTR   = Pointer auf Anfang des Puffer *
*  Return-Wert       : keiner                                 *
*  Info              : Der Aufrufer ist dafür verantwortlich, *
*                      daß die Größe des übergebenen Puffers  *
*                      zur Speicherung des Bereichs ausreicht.*
************************************************************/

void VioGet( BYTE x1, BYTE y1, BYTE x2,
             BYTE y2, VEL far * bptr )
{
 int      nbytes;          /* zu kopierende Bytes pro Zeile */
 VPFAR    vioptr;               /* Pointer in den Video-RAM */
 unsigned temp;

 nbytes = ( x2 - x1 + 1 ) * VELLEN;      /* Bytes pro Zeile */
 for ( ; y1 <= y2; ++y1 )            /* die Zeilen durchlaufen */
  {           /* jeweils eine Zeile in den Puffer holen */
   vioptr = VPOS(x1, y1);   /* Pointer auf 1. Byte in Zeile */
   MOVE( vioptr, bptr, nbytes);
   (BYTE far *) bptr += nbytes;
  }
}


/************************************************************
*  Funktion          : V i o P u t                            *
**---------------------------------------------------------**
*  Aufgabe           : Kopiert den Inhalt eines Puffers in    *
*                      einen Bildschirmbereich.               *
*  Eingabe-Parameter: X1, Y1 = obere linke Ecke des Bereichs *
*                      X2, Y2 = untere rechte Ecke des Ber.   *
*                      BPTR   = Pointer auf Anfang des Puffer *
*  Return-Wert       : keiner                                 *
************************************************************/
```

Listing 2: *(Fortsetzung)*

```
void VioPut( BYTE x1, BYTE y1, BYTE x2,
             BYTE y2, VEL far * bptr )
{
 int      nbytes;          /* zu kopierende Bytes pro Zeile */
 VPFAR    vioptr;               /* Pointer in den Video-RAM */
 unsigned temp;

 nbytes = ( x2 - x1 + 1 ) * VELLEN;      /* Bytes pro Zeile */
 for ( ; y1 <= y2; ++y1 )            /* die Zeilen durchlaufen */
  {          /* jeweils eine Zeile in den Video-RAM kopieren */
   vioptr = VPOS(x1, y1);   /* Pointer auf 1. Byte in Zeile */
   MOVE( bptr, vioptr, nbytes);
   (BYTE far *) bptr += nbytes;
  }
}


/************************************************************
*  Funktion          : V i o W i n O p e n                    *
**---------------------------------------------------------**
*  Aufgabe           : Öffnet ein Bildschirmfenster und       *
*                      sichert den Inhalt des darunter-       *
*                      liegenden Bildschirmbereichs.          *
*  Eingabe-Parameter: X1, Y1 = obere linke Ecke des Bereichs *
*                      X2, Y2 = untere rechte Ecke des Ber.   *
*  Return-Wert       : TRUE, wenn das Fenster geöffnet werden *
*                      konnte, sonst FALSE                    *
************************************************************/

BYTE VioWinOpen(  BYTE x1, BYTE y1, BYTE x2, BYTE y2 )
{
 VEL * bptr;        /* Ptr. auf Puffer mit Bildschirminhalts */
 WIPTR wptr;  /* Pointer auf Vektor mit Fensterbeschreibern */

 if ( (bptr = (VEL *) malloc(BUFLEN( x1, y1, x2, y2 ))) )
  {                    /* es konnte ein Puffer allokiert werden */
   wptr = (WIPTR) realloc(winptr, WINDESLEN * (winopen+1));
   if ( wptr )      /* konnte der Vektor vergrößert werden? */
    {                                                  /* Ja */
     winptr = wptr;  /* die neue Adresse des Vektors merken */
     (wptr += winopen)->x1 = x1;          /* die Koordinaten */
     wptr->x2 = x2;                       /* des Fensters im */
     wptr->y1 = y1;                       /* Fensterbeschr.  */
     wptr->y2 = y2;                       /* merken          */
     wptr->curs = vspalte;    /* die aktuelle Cursorposition */
     wptr->curz = vzeile;     /* ebenfalls speichern         */
     wptr->viewx1 = viewx1;            /* die Koordinaten des */
     wptr->viewy1 = viewy1;            /* View-Breiches im    */
     wptr->viewx2 = viewx2;            /* Fensterbeschreiber  */
     wptr->viewy2 = viewy2;            /* speichern           */
     VioSetView( x1, y1, x2, y2 );     /* neuer View-Bereich */
     VioGet( x1, y1, x2, y2, wptr->winmem = bptr );
     ++winopen;              /* Anzahl Fenster inkrementieren */
     return TRUE;           /* Fenster konnte geöffnet werden */
    }
   else         /* der Vektor konnte nicht vergrößert werden */
    {
     free( bptr );          /* Bildschirmpuffer wieder freig. */
     return FALSE;                       /* mit Fehler zurück */
    }
  }
 else              /* es konnte kein Puffer allokiert werden */
  return FALSE;
}


/************************************************************
*  Funktion          : V i o W i n C l o s e                  *
**---------------------------------------------------------**
*  Aufgabe           : Schließt das zuletzt geöffnete Fenster *
*                      wieder.                                *
*  Eingabe-Parameter: REDRAW = TRUE: der Bildschirminhalt    *
*                                    unter dem Fenster wird   *
*                                    zurückkopiert.           *
*  Return-Wert       : keiner                                 *
************************************************************/
```

Listing 2: *(Fortsetzung)*

```
void VioWinClose( BYTE redraw )
{
 WIPTR wptr;  /* Pointer auf Vektor mit den Fenster-Beschr. */
 union vel * buptr;   /* Pointer auf Puffer für das Fenster */
 BYTE  i, j,                                /* Schleifenzähler */
       y1, y2, x1;                    /* Koordinaten des Fensters */
 VPFAR video;                         /* Pointer in den Video-RAM */

 if (winopen)         /* ist überhaupt ein Fenster geöffnet? */
  {                                                    /* Ja */
   wptr = winptr + winopen - 1;   /* Ptr auf Fensterbeschr. */
   if ( redraw )     /* alten Fensterinhalt zurückkopieren? */
    {               /* Ja, aus Puffer in Video-RAM kopieren */
     VioPut( wptr->x1, wptr->y1, wptr->x2,
             wptr->y2, wptr->winmem );
     VioSetView( wptr->viewx1, wptr->viewy1, wptr->viewx2,
                 wptr->viewy2 );     /* View-Bereich zurück */
     VioSetCursor( wptr->curs, wptr->curz );       /* Cursor */
    }
   free( (void *) wptr->winmem );   /* Fenstersp. freigeben */
   /*-- den Vektor mit den Fensterbeschr. verkleinern ------*/
   winptr = (WIPTR) realloc(winptr, WINDESLEN*(--winopen)+1);
  }
}

/*************************************************************
*  Funktion           : V i o S e t C u r s o r               *
**-------------------------------------------------------------**
*  Aufgabe            : Setzt den blinkenden Bildschirmcursor  *
*                       und die interne Ausgabeposition.       *
*  Eingabe-Parameter: SPALTE = die neue Cursorposition         *
*                       ZEILE                                  *
*  Return-Wert        : keiner                                 *
*  Info               :                                        *
*                                                              *
*************************************************************/

void VioSetCursor( BYTE spalte, BYTE zeile )
{
 union REGS regs;      /* Prozessorregs. für Interruptaufruf */

 regs.h.ah = 2;              /* Funktionsnummer für Set Cursor */
 regs.h.bh = 0;         /* auf die Bildschirmseite 0 zugreifen */
 regs.h.dh = vzeile = zeile;                  /* Zeile merken */
 regs.h.dl = vspalte = spalte;               /* Spalte merken */
 int86(0x10, &regs, &regs);      /* BIOS-Video-Intr. aufrufen */
}

/*************************************************************
*  Funktion           : V i o P r i n t                       *
**-------------------------------------------------------------**
*  Aufgabe            : Schreibt einen String direkt in den    *
*                       Video-RAM.                             *
*  Eingabe-Parameter: SPALTE = die Ausgabeposition             *
*                       ZEILE                                  *
*                       FARBE  = das Attribut für die Zeichen  *
*                       CURSOR = TRUE, wenn der Cursor hinter  *
*                                die Ausgabe gesetzt wird      *
*                       STRING = Pointer auf den String        *
*  Return-Wert        : keiner                                 *
*  Info               : - Der String wird unformatiert ausge-  *
*                           geben.                             *
*************************************************************/

void VioPrint( BYTE spalte, BYTE zeile, BYTE farbe,
               BYTE cursor, char * string )

{
 register VPFAR lptr;  /* Laufzeiger zum Schreiben des Str. */
 int neupos;                        /* Cursorposition als Offset */

 neupos = spalte + zeile * anzcol;
 lptr = VPOS(spalte, zeile); /* Pointer in Video-RAM setzen */
 for ( ; *string ; ++neupos, ++lptr)        /* String durchl. */
  {
   lptr->h.zeichen = *(string++);   /* Zeichen in VIDEO-RAM */
   lptr->h.attribut = farbe; /* Attribut des Zeichen setzen */
  }
```

Listing 2: *(Fortsetzung)*

```
 if ( cursor )          /* Cursor hinter die Ausgabe setzen? */
  VioSetCursor( neupos % anzcol , neupos / anzcol );  /* Ja */
}

/*************************************************************
*  Funktion           : V i o P r i n t f                     *
**-------------------------------------------------------------**
*  Aufgabe            : Formatiert einen String wie bei PRINTF *
*                       und schreibt ihn dann direkt in den    *
*                       Video-RAM.                             *
*  Eingabe-Parameter: SPALTE = die Ausgabeposition             *
*                       ZEILE                                  *
*                       FARBE  = das Attribut für die Zeichen  *
*                       CURSOR = TRUE, wenn der Cursor hinter  *
*                                die Ausgabe gesetzt wird      *
*                       STRING = Pointer auf den String        *
*                       ...    = weitere Argumente             *
*  Return-Wert        : keiner                                 *
*  Info               : - Der String kann die Formatkenn-      *
*                           zeichen wie bei PRINTF enthalten.  *
*                         - Der Cursor wird nach der Ausgabe   *
*                           hinter das letzte Zeichen gesetzt. *
*************************************************************/

void VioPrintf( BYTE spalte, BYTE zeile, BYTE farbe,
                BYTE cursor, char * string, ... )

{
 va_list parameter;     /* Parameter-Liste für VA_... Macros */
 char ausgabe[255];          /* Puffer für formatierten String */
 int neupos;                  /* neu Cursorposition als Offset */

 va_start( parameter, string );        /* Parameter umwandeln */
 vsprintf( ausgabe, string, parameter );       /* formatieren */
 VioPrint( spalte, zeile, farbe, cursor, ausgabe );
}

/*************************************************************
*  Funktion           : V i o S t r e p                       *
**-------------------------------------------------------------**
*  Aufgabe            : Baut einen String aus einem einheit-   *
*                       lichen Zeichen auf.                    *
*  Eingabe-Parameter: ASCII = Das Zeichen                      *
*                       ANZ   = Anzahl der Wiederholungen      *
*  Return-Wert        : Pointer auf den erstellten String      *
*  Info               : - ANZAHL darf nicht größer als 132     *
*                           sein.                              *
*                         - Der String "lebt" nur bis zum      *
*                           nächsten Aufruf von VioStrep.      *
*************************************************************/

char *VioStrep( char zeichen, BYTE anz )

{
 static char buf[133];        /* Puffer zum Aufbau des String */

 /*-- Puffer mit Zeichen füllen und durch NUL abschließen --*/

 memset( buf, zeichen, anz = ((anz > 132) ? 132 : anz) );
 buf[anz] = '\0';                       /* das String-Ende setzen */
 return buf;          /* Pointer auf den Puffer zurückliefern */
}

/*************************************************************
*  Funktion           : V i o F i l l                         *
**-------------------------------------------------------------**
*  Aufgabe            : Einen Bildschirmbereich mit einem      *
*                       Zeichen füllen.                        *
*  Eingabe-Parameter: X1, Y1 = obere linke Ecke des Bereichs   *
*                       X2, Y2 = untere rechte Ecke des Ber.   *
*                       ZEICHEN= das Füll-Zeichen              *
*                       FARBE  = Attribut für das Füll-Zeichen *
*  Return-Wert        : keiner                                 *
*  Info               :                                        *
*                                                              *
*************************************************************/
```

Listing 2: *(Fortsetzung)*

```
void VioFill( BYTE x1, BYTE y1, BYTE x2,
              BYTE y2, char zeichen, BYTE farbe)

{
 char * line;          /* Pointer auf eine auszugebende Zeile */

 line = VioStrep( zeichen, x2-x1+1 );        /* Zeile aufbauen */
 for ( ; y1 <= y2; ++y1)                /* Zeilen durchlaufen */
  VioPrint( x1, y1, farbe, FALSE, line ); /* Zeile ausgeben */
}

/***********************************************************
*  Funktion          : V i o F r a m e                      *
**--------------------------------------------------------**
*  Aufgabe           : Zieht einen Rahmen um einen Bild-    *
*                      schirmbereich.                       *
*  Eingabe-Parameter: X1, Y1 = obere linke Ecke des Bereichs *
*                      X2, Y2 = untere rechte Ecke des Ber.  *
*                      RAHMEN = der Rahmen-Typ               *
*                      FARBE  = Farbe des Rahmen             *
*  Return-Wert       : keiner                               *
*  Info              : - RAHMEN muß eine der folgenden Kon-  *
*                        stanten sein: EINRA, DOPRA, VOLLRA, *
*                        PUNKTRA                            *
*                      - Verändert den VioStrep-Puffer!      *
***********************************************************/

void VioFrame( BYTE x1, BYTE y1, BYTE x2,
               BYTE y2, BYTE rahmen, BYTE farbe )

{
 static char rahz[4][6] = {     /* die versch. Rahmenzeichen */
                          { '┌', '┐', '└', '┘', '│', '─' },
                          { '╔', '╗', '╚', '╝', '║', '═' },
                          { '█', '█', '█', '█', '█', '█' },
                          { '█', '█', '█', '█', '█', '█' }
                         };

 BYTE i, k;                              /* Schleifenzähler */
 char *strepu,               /* Pointer auf den VioStrep-Puffer */
      senkr;                            /* senkrechter Strich */

 VioPrintf( x1, y1, farbe, FALSE, "%c", rahz[ rahmen ][ 0 ] );
 VioPrint( x1+1, y1, farbe, FALSE,
  strepu = VioStrep( rahz[ rahmen ][ 5 ], x2-x1-1 ) );
 VioPrintf( x2, y1, farbe, FALSE, "%c", rahz[ rahmen ][ 1 ] );
 for (senkr=rahz[ rahmen ][ 4 ], i=y1+1, k=y2-1; i <= k; ++i)
  {
   VioPrintf( x1, i, farbe, FALSE, "%c", senkr );
   VioPrintf( x2, i , farbe, FALSE, "%c", senkr );
  }
 VioPrintf( x1, y2, farbe, FALSE, "%c", rahz[ rahmen ][ 2 ] );
 VioPrint( x1+1, y2, farbe, FALSE, strepu );
 VioPrintf( x2, y2, farbe, FALSE, "%c", rahz[ rahmen ][ 3 ] );
}

/***********************************************************
*  Funktion          : V i o S c r o l l U p                *
**--------------------------------------------------------**
*  Aufgabe           : Scrollt einen Bildschirmbereich um   *
*                      eine oder mehrere Zeilen nach oben.  *
*  Eingabe-Parameter: X1, Y1 = obere linke Ecke des Bereichs *
*                      X2, Y2 = untere rechte Ecke des Ber.  *
*                      ANZAHL = Anzahl der Zeilen, um die der *
*                               Bereich gescrollt werden soll *
*                      FARBE  = Farbe für die Leerzeilen     *
*                               (255 : kein Löschen d. Leerz.)*
*  Return-Wert       : keiner                               *
***********************************************************/

void VioScrollUp( BYTE x1, BYTE y1, BYTE x2,
                  BYTE y2, BYTE anzahl, BYTE farbe )
{
 VPFAR alt,                                   /* Laufzeiger */
       neu;
 int   anzb;                      /* Anzahl Bytes pro Zeile */
 BYTE  anzy,                      /* die Anzahl der Zeilen */
       zeile;                        /* die aktuelle Zeile */
```

Listing 2: *(Fortsetzung)*

```
 anzb = (x2 - x1 + 1) * VELLEN;                /* Anzahl Bytes */
 anzy = y2 - (zeile = y1) + 1;                /* Anzahl Zeilen */

 for ( ; anzy ; --anzy, ++zeile)      /* Zeilen durchlaufen */
  { /* Pointer auf Anfang der zu übertragenen Zeile setzen */
   alt = VPOS(x1, zeile);
   neu = VPOS(x1, zeile-anzahl);
   MOVE( alt, neu, anzb );                  /* Zeile kopieren */
  }
 if ( farbe != 255 )                   /* Leerzeilen löschen? */
  VioClear( x1, y2+1-anzahl, x2, y2, farbe);             /* Ja */
}

/***********************************************************
*  Funktion          : V i o S c r o l l D o w n            *
**--------------------------------------------------------**
*  Aufgabe           : Scrollt einen Bildschirmbereich um   *
*                      eine oder mehrere Zeilen nach unten. *
*  Eingabe-Parameter: X1, Y1 = obere linke Ecke des Bereichs *
*                      X2, Y2 = untere rechte Ecke des Ber.  *
*                      ANZAHL = Anzahl der Zeilen, um die der *
*                               Bereich gescrollt werden soll *
*                      FARBE  = Farbe für die Leerzeilen     *
*                               (255 : kein Löschen d. Leerz.)*
*  Return-Wert       : keiner                               *
***********************************************************/

void VioScrollDown( BYTE x1, BYTE y1, BYTE x2,
                    BYTE y2, BYTE anzahl, BYTE farbe )

{
 VPFAR alt,                                   /* Laufzeiger */
       neu;
 int   anzb;                      /* Anzahl Bytes pro Zeile */
 BYTE  anzy,                      /* die Anzahl der Zeilen */
       zeile;                        /* die aktuelle Zeile */

 anzb = (x2 - x1 + 1) * VELLEN;                /* Anzahl Bytes */
 anzy = (zeile = y2) - y1 + 1;     /* Anzahl Zeilen berechnen */

 for ( ; anzy ; --anzy, --zeile)      /* Zeilen durchlaufen */
  { /* Pointer auf Anfang der zu übertragenen Zeile setzen */
   alt = VPOS(x1, zeile);
   neu = VPOS(x1, zeile+anzahl);
   MOVE( alt, neu, anzb );                  /* Zeile kopieren */
  }
 if ( farbe != 255 )                   /* Leerzeilen löschen? */
  VioClear( x1, y1, x2, y1+anzahl-1, farbe);             /* Ja */
}

/***********************************************************
*  Funktion          : V i o S c r o l l H o r i            *
**--------------------------------------------------------**
*  Aufgabe           : Scrollt einen Bildschirmbereich um   *
*                      eine oder mehrere Spalten nach links *
*                      oder rechts.                         *
*  Eingabe-Parameter: X1, Y1 = obere linke Ecke des Bereichs *
*                      X2, Y2 = untere rechte Ecke des Ber.  *
*                      ANZAHL = Anzahl der Spalten, um die   *
*                               der B. gescrollt werden soll *
*                      FARBE  = Farbe für die Leerspalten    *
*                      LEFT   = TRUE, wenn der Bereich nach  *
*                               Links gescrollt werden soll. *
*                               (255 : kein Löschen d. Leers.)*
*  Return-Wert       : keiner                               *
***********************************************************/

void VioScrollHori( BYTE x1, BYTE y1, BYTE x2, BYTE y2,
                    BYTE anzahl, BYTE farbe, BYTE left )

{
 VEL   zbuf[ 132 ];           /* nimmt die jeweilige Zeile auf */
 VPFAR alt,                                   /* Laufzeiger */
       neu,
       zptr = zbuf;              /* Pointer auf den Zeilenpuffer */
 int   anzb;                      /* Anzahl Bytes pro Zeile */
 BYTE  anzy,                      /* die Anzahl der Zeilen */
       zeile;                        /* die aktuelle Zeile */
 int   off;                        /* Entfernung ALT --> NEU */
```

Listing 2: *(Fortsetzung)*

```
anzb = (x2 - x1 + 1) * VELLEN;                       /* Anzahl Bytes */
anzy = y2 - (zeile = y1) + 1;                       /* Anzahl Zeilen */
off = ( left ) ? -anzahl : anzahl;          /* Entfernung setzen */
zptr = zbuf;

for ( ; anzy ; --anzy, ++zeile)          /* Zeilen durchlaufen */
 { /* Pointer auf Anfang der zu übertragenen Zeile setzen */
  neu = ( alt = VPOS(x1, zeile) ) + off;

  #ifndef NEARDATA           /* in "kleinem" Speichermodell? */
   MOVE( alt, neu, anzb );          /* Nein, Spalten kopieren */
  #else                                              /* Ja */
    if ( left )                        /* nach links scrollen? */
      MOVE( alt, neu, anzb );          /* Ja, Spalten kopieren */
    else                               /* Nein, nach rechts */
     {    /* MOVEDATA kopiert immer von links --> rechts */
      MOVE( alt, zptr, anzb );         /* erst in Puffer holen */
      MOVE( zptr, neu, anzb );       /* von da wieder auf Bs. */
     }
  #endif

 }
/*-- frei gewordene Spalten löschen ----------------------*/

if ( farbe != 255 )                    /* Leerspalten löschen? */
 if ( left )                         /* Ja, nach Links scrollen? */
  VioClear( x2-anzahl+1, y1, x2, y2, farbe);
 else                              /* Nein , nach rechts scrollen */
  VioClear( x1, y1, x1+anzahl-1, y2, farbe);
}


/***************************************************************
*  Funktion          : V i o M o v e U p                          *
**------------------------------------------------------------**
*  Aufgabe           : Schiebt das aktuelle Fenster um eine      *
*                      oder mehrere Zeilen nach oben.             *
*  Eingabe-Parameter: ANZAHL = Anzahl der Zeilen                  *
*  Return-Wert       : TRUE, wenn das Fenster verschoben          *
*                      werden konnte, sonst FALSE                 *
*  Info              : - Der Aufrufer trägt dafür Verant-         *
*                        wortung, daß das Fenster nicht über      *
*                        den Bildschirm hinaus verschoben         *
*                        wird.                                    *
*                      - Der View-Bereich wird mit dem Fen-       *
*                        ster nach oben verschoben.               *
*                      - Befindet sich der Cursor innerhalb       *
*                        des Fensters, wird auch er ver-          *
*                        schoben.                                 *
***************************************************************/

BYTE VioMoveUp( BYTE anzahl )
{
 WIPTR wptr;          /* Pointer auf aktuellen Fenster-Beschr. */
 VEL * bptr,  /* Ptr. auf Puffer für überschriebenen B.ber. */
     * aktbuf;                          /* aktueller Fensterpuffer */
 BYTE  x1, y1,       /* die Koordinaten des aktuellen Fenster */
       x2, y2;
 int   zlen;                        /* Länge des Zwischenspeichers */

 if ( winopen )         /* ist überhaupt ein Fenster geöffnet? */
  {                                                      /* Ja */
  wptr = winptr + winopen - 1;    /* Ptr auf Fensterbeschr. */
  bptr = (VEL *) malloc( zlen = BUFLEN(wptr->x1,
                         1, wptr->x2, anzahl ) );
  if ( bptr )  /* konnte ein Zwischensp. allokiert werden? */
   {                                                     /* Ja */
    /*-- 1. Zeilen über Fenster in Zwischenspeicher holen -*/
    VioGet(x1 = wptr->x1, y1 - anzahl,
           x2 = wptr->x2, ( y1 = wptr->y1 ) -1, bptr );

    /*-- 2. Fenster um ANZAHL Zeilen nach oben scrollen ---*/
    VioScrollUp( x1, y1, x2, y2=wptr->y2, anzahl, NOCLEAR );

    /*-- 3. untere Zeilen des Fensters zurück auf Bilds. --*/
    VioPut( x1, y2-anzahl+1, x2, y2, (aktbuf=wptr->winmem) +
            (x2-x1+1) * (y2-y1-anzahl+1) );
```

Listing 2: *(Fortsetzung)*

```
    /*-- 4. frei gewordene Zeilen aus Puffer entfernen ----*/
    memmove( aktbuf + (x2-x1+1) * anzahl, aktbuf,
             BUFLEN(x1, y1, x2, y2-anzahl ) );

    /*-- 5. neu überdeckte Zeilen in Puffer kopieren ------*/
    memmove( aktbuf, bptr, zlen );

    /*-- 6. Cursor versetzen ------------------------------*/
    if ( (x1 <= vspalte ) && (x2 >= vspalte ) &&
         (y1 <= vzeile ) && (y2 >= vzeile ) )
      VioSetCursor( vspalte , vzeile - anzahl );

    /*-- 7. Koordinaten im Fensterbeschreiber anpassen ----*/
    wptr->y1 -= anzahl;
    wptr->y2 -= anzahl;
    viewy1 -= anzahl;
    viewy2 -= anzahl;

    free( bptr );        /* Zwischenspeicher wieder freigeben */
    return TRUE;                                  /* geschafft! */
   }
  else /* es konnte kein Zwischenspeicher allokiert werden */
   return FALSE;                          /* mit Fehler zurück */
  }
 else
  return FALSE;                 /* es ist kein Fenster geöffnet */
}


/***************************************************************
*  Funktion          : V i o M o v e D o w n                      *
**------------------------------------------------------------**
*  Aufgabe           : Schiebt das aktuelle Fenster um eine      *
*                      oder mehrere Zeilen nach unten.            *
*  Eingabe-Parameter: ANZAHL = Anzahl der Zeilen                  *
*  Return-Wert       : TRUE, wenn das Fenster verschoben          *
*                      werden konnte, sonst FALSE                 *
*  Info              : - Der Aufrufer trägt dafür Verant-         *
*                        wortung, daß das Fenster nicht über      *
*                        den Bildschirm hinaus verschoben         *
*                        wird.                                    *
*                      - Der View-Bereich wird mit dem Fen-       *
*                        ster nach unten verschoben.              *
*                      - Befindet sich der Cursor innerhalb       *
*                        des Fensters, wird auch er ver-          *
*                        schoben.                                 *
***************************************************************/

BYTE VioMoveDown( BYTE anzahl )
{
 WIPTR wptr;          /* Pointer auf aktuellen Fenster-Beschr. */
 VEL * bptr,  /* Ptr. auf Puffer für überschriebenen B.ber. */
     * aktbuf;                          /* aktueller Fensterpuffer */
 BYTE  x1, y1,       /* die Koordinaten des aktuellen Fenster */
       x2, y2;
 int   zlen;                        /* Länge des Zwischenspeichers */

 if ( winopen )         /* ist überhaupt ein Fenster geöffnet? */
  {                                                      /* Ja */
  wptr = winptr + winopen - 1;    /* Ptr auf Fensterbeschr. */
  bptr = (VEL *) malloc( zlen = BUFLEN(wptr->x1,
                         1, wptr->x2, anzahl ) );
  if ( bptr )  /* konnte ein Zwischensp. allokiert werden? */
   {                                                     /* Ja */
    /*-- 1. Zeilen unter Fenster in Zwischenspeicher holen */
    VioGet(x1 = wptr->x1, y2+1,
           x2 = wptr->x2, ( y2 = wptr->y2 ) + anzahl, bptr );

    /*-- 2. Fenster um ANZAHL Zeilen nach unten scrollen --*/
    VioScrollDown(x1, y1=wptr->y1, x2, y2, anzahl, NOCLEAR );

    /*-- 3. obere Zeilen des Fensters zurück auf Bilds. ---*/
    VioPut( x1, y1, x2, y1+anzahl-1, (aktbuf=wptr->winmem) );

    /*-- 4. frei gewordene Zeilen aus Puffer entfernen ----*/
    memmove( aktbuf, aktbuf + (x2-x1+1) * anzahl,
             BUFLEN(x1, y1, x2, y2-anzahl ) );
```

Listing 2: *(Fortsetzung)*

```
    /*-- 5. neu überdeckte Zeilen in Puffer kopieren ------*/
    memmove( (BYTE *) aktbuf + BUFLEN(x1, y1, x2, y2-anzahl),
             bptr, zlen );

    /*-- 6. Cursor versetzen -------------------------------*/
    if ( (x1 <= vspalte ) && (x2 >= vspalte ) &&
         (y1 <= vzeile ) && (y2 >= vzeile ) )
      VioSetCursor( vspalte, vzeile + anzahl );

    /*-- 7. Koordinaten im Fensterbeschreiber anpassen ----*/
    wptr->y1 += anzahl;
    wptr->y2 += anzahl;
    viewy1 += anzahl;
    viewy2 += anzahl;

    free( bptr );       /* Zwischenspeicher wieder freigeben */
    return TRUE;                              /* geschafft! */
   }
  else /* es konnte kein Zwischenspeicher allokiert werden */
   return FALSE;                       /* mit Fehler zurück */
 }
 else
  return FALSE;                /* es ist kein Fenster geöffnet */
}

/************************************************************
* Funktion        : V i o M o v e R i g h t                *
**---------------------------------------------------------**
* Aufgabe         : Schiebt das aktuelle Fenster um eine    *
*                   oder mehrere Spalten nach rechts.       *
* Eingabe-Parameter: ANZAHL = Anzahl der Spalten            *
* Return-Wert     : TRUE, wenn das Fenster verschoben       *
*                   werden konnte, sonst FALSE              *
* Info            : - Der Aufrufer trägt dafür Verant-      *
*                     wortung, daß das Fenster nicht über   *
*                     den Bildschirm hinaus verschoben      *
*                     wird.                                 *
*                   - Der View-Bereich wird mit dem Fen-    *
*                     ster nach rechts verschoben.          *
*                   - Befindet sich der Cursor innerhalb    *
*                     des Fensters, wird auch er ver-       *
*                     schoben.                              *
************************************************************/

BYTE VioMoveRight( BYTE anzahl )
{
 WIPTR wptr;        /* Pointer auf aktuellen Fenster-Beschr. */
 VEL * bptr,  /* Ptr. auf Puffer für überschriebenen B.ber. */
     * aktbuf,                     /* aktueller Fensterpuffer */
     * lptr,                 /* Laufzeiger in Fensterpuffer */
     * lbptr;            /* Laufzeiger in Zwischenspeicher */
 BYTE  x1, y1,      /* die Koordinaten des aktuellen Fenster */
       x2, y2,
       off,          /* Offsetwert zum Durchlaufen des Puffers */
       i, j;                             /* Schleifenzähler */
 int   zlen;                  /* Länge des Zwischenspeichers */

 if ( winopen )       /* ist überhaupt ein Fenster geöffnet? */
  {                                                /* Ja */
   wptr = winptr + winopen - 1;   /* Ptr auf Fensterbeschr. */
   bptr = (VEL *) malloc( zlen = BUFLEN(1, wptr->y1,
                                 anzahl, wptr->y2 ) );
   if ( bptr )  /* konnte ein Zwischensp. allokiert werden? */
    {                                              /* Ja */
    /*-- 1. Spalten neben Fenster in Zwischensp. holen ----*/
    VioGet( x2 + 1, y1 = wptr->y1, (x2 = wptr->x2) + anzahl,
            y2 = wptr->y2, bptr );

    /*-- 2. Fenster um ANZAHL Spalten n. rechts scrollen --*/
    VioScrollRight( x1 = wptr->x1, y1, x2, y2,
                    anzahl, NOCLEAR );

    /*-- 3. linke Spalten des Fensters zurück auf Bilds. --*/
    lptr = aktbuf= wptr->winmem;   /* Ptr auf Pufferanfang */
    off = ( x2 - x1 + 1 );       /* Länge einer Zeile in VEL */
    j = x1 + anzahl - 1;         /* rechte Spalte für VioPut */
    for ( i = y1; i <= y2; ++i, lptr += off )
     VioPut( x1, i, j, i, lptr );      /* eine Zeile zurück */
```

Listing 2: *(Fortsetzung)*

```
    /*-- 4. frei gewordene Spalten aus Puffer entfernen ---*/
    lptr = aktbuf;                /* Ptr auf Fensterpuffer */
    j = (off - anzahl) * VELLEN;       /* zu versch. Bytes */
    for ( i = y1; i <= y2; ++i, lptr += off )
     memmove( lptr, lptr + anzahl, j );

    /*-- 5. Cursor versetzen ------------------------------*/
    if ( (x1 <= vspalte ) && (x2 >= vspalte ) &&
         (y1 <= vzeile ) && (y2 >= vzeile ) )
      VioSetCursor( vspalte + anzahl , vzeile );

    /*-- 6. neu überdeckte Zeilen in Puffer kopieren ------*/
    lptr = aktbuf + (x2-x1+1-anzahl) ; /* Ptr auf F.puffer */
    lbptr = bptr;                 /* Ptr auf Zwischenpuffer */
    j = anzahl * VELLEN;          /* zu verschiebende Bytes */
    for (  ; y1 <= y2; ++y1, lptr += off, lbptr += anzahl )
     memmove( lptr, lbptr, j );

    /*--7. Koordinaten im Fensterbeschreiber anpassen -----*/
    wptr->x1 += anzahl;
    wptr->x2 += anzahl;
    viewx1 += anzahl;
    viewx2 += anzahl;

    free( bptr );       /* Zwischenspeicher wieder freigeben */
    return TRUE;                              /* geschafft! */
   }
  else /* es konnte kein Zwischenspeicher allokiert werden */
   return FALSE;                       /* mit Fehler zurück */
 }
 else
  return FALSE;                /* es ist kein Fenster geöffnet */
}


/************************************************************
* Funktion        : V i o M o v e L e f t                  *
**---------------------------------------------------------**
* Aufgabe         : Schiebt das aktuelle Fenster um eine    *
*                   oder mehrere Spalten nach links.        *
* Eingabe-Parameter: ANZAHL = Anzahl der Spalten            *
* Return-Wert     : TRUE, wenn das Fenster verschoben       *
*                   werden konnte, sonst FALSE              *
* Info            : - Der Aufrufer trägt dafür Verant-      *
*                     wortung, daß das Fenster nicht über   *
*                     den Bildschirm hinaus verschoben      *
*                     wird.                                 *
*                   - Der View-Bereich wird mit dem Fen-    *
*                     ster nach links verschoben.           *
*                   - Befindet sich der Cursor innerhalb    *
*                     des Fensters, wird auch er ver-       *
*                     schoben.                              *
************************************************************/

BYTE VioMoveLeft( BYTE anzahl )
{
 WIPTR wptr;        /* Pointer auf aktuellen Fenster-Beschr. */
 VEL * bptr,  /* Ptr. auf Puffer für überschriebenen B.ber. */
     * aktbuf,                     /* aktueller Fensterpuffer */
     * lptr,                 /* Laufzeiger in Fensterpuffer */
     * lbptr;            /* Laufzeiger in Zwischenspeicher */
 BYTE  x1, y1,      /* die Koordinaten des aktuellen Fenster */
       x2, y2,
       off,          /* Offsetwert zum Durchlaufen des Puffers */
       i, j;                             /* Schleifenzähler */
 int   zlen;                  /* Länge des Zwischenspeichers */

 if ( winopen )       /* ist überhaupt ein Fenster geöffnet? */
  {                                                /* Ja */
   wptr = winptr + winopen - 1;   /* Ptr auf Fensterbeschr. */
   bptr = (VEL *) malloc( zlen = BUFLEN(1, wptr->y1,
                                 anzahl, wptr->y2 ) );
   if ( bptr )  /* konnte ein Zwischensp. allokiert werden? */
    {                                              /* Ja */
    /*-- 1. Spalten neben Fenster in Zwischensp. holen ----*/
    VioGet( x1 - anzahl, y1 = wptr->y1, (x1 = wptr->x1) - 1,
            y2 = wptr->y2, bptr );
```

Listing 2: *(Fortsetzung)*

```
     /*-- 2. Fenster um ANZAHL Spalten nach links scrollen -*/
     VioScrollLeft( x1, y1, x2 = wptr->x2, y2,
                    anzahl, NOCLEAR );

     /*-- 3. rechte Spalten des Fensters zurück auf Bilds. -*/
     off = ( x2 - x1 + 1 );        /* Länge einer Zeile in VEL */
     lptr = (aktbuf = wptr->winmem) + off - anzahl;
     j = x2 - anzahl + 1;           /* linke Spalte für VioPut */
     for ( i = y1; i <= y2; ++i, lptr += off )
      VioPut( j, i, x2, i, lptr );        /* eine Zeile zurück */

     /*-- 4. frei gewordene Spalten aus Puffer entfernen ---*/
     lptr = aktbuf;                   /* Ptr auf Fensterpuffer */
     j = (off - anzahl) * VELLEN;       /* zu versch. Bytes */
     for ( i = y1; i <= y2; ++i, lptr += off )
      memmove( lptr + anzahl, lptr, j );


     /*-- 5. Cursor versetzen ------------------------------*/
     if ( (x1 <= vspalte ) && (x2 >= vspalte ) &&
          (y1 <= vzeile ) && (y2 >= vzeile ) )
      VioSetCursor( vspalte - anzahl , vzeile );

     /*-- 6. neu überdeckte Zeilen in Puffer kopieren ------*/
     lptr = aktbuf;                   /* Ptr auf Fensterpuffer */
     lbptr = bptr;                   /* Ptr auf Zwischenpuffer */
     j = anzahl * VELLEN;            /* zu verschiebende Bytes */
     for (  ; y1 <= y2; ++y1, lptr += off, lbptr += anzahl )
      memmove( lptr, lbptr, j );

     /*-- 7. Koordinaten im Fensterbeschreiber anpassen ----*/
     wptr->x1 -= anzahl;
     wptr->x2 -= anzahl;
     viewx1 -= anzahl;
     viewx2 -= anzahl;

     free( bptr );        /* Zwischenspeicher wieder freigeben */
     return TRUE;                               /* geschafft! */
    }
   else /* es konnte kein Zwischenspeicher allokiert werden */
    return FALSE;                     /* mit Fehler zurück */
  }
 else
  return FALSE;               /* es ist kein Fenster geöffnet */
}


/***************************************************************
* Funktion          : V i o S e t W i n                       *
**-----------------------------------------------------------**
* Aufgabe           : Schiebt das aktuelle Fenster an eine    *
*                     neue Bildschirmposition.                *
* Eingabe-Parameter: XNEU,  = neue Koordinate der oberen      *
*                     YNEU     linken Fensterecke             *
* Return-Wert       : TRUE, wenn das Fenster verschoben       *
*                     werden konnte, sonst FALSE              *
* Info              : - Der Aufrufer trägt dafür Verant-      *
*                       wortung, daß das Fenster nicht über   *
*                       den Bildschirm hinaus verschoben      *
*                       wird.                                 *
*                     - Der View-Bereich wird mit dem Fen-    *
*                       ster verschoben.                      *
*                     - Befindet sich der Cursor innerhalb    *
*                       des Fensters, wird auch er ver-       *
*                       schoben.                              *
***************************************************************/

BYTE VioSetWin( BYTE xneu, BYTE yneu )
{
 WIPTR wptr;          /* Pointer auf aktuellen Fenster-Beschr. */
 VEL * bptr,   /* Ptr. auf Puffer für überschriebenen B.ber. */
     * aktbuf;                     /* aktueller Fensterpuffer */
 BYTE  x1, y1,       /* die Koordinaten des aktuellen Fenster */
       x2, y2;
 int   zlen,                   /* Länge des Zwischenspeichers */
       xdif,              /* Entfernung des neuen vom alten */
       ydif;              /* Fenster in Spalten und Zeilen */
```

Listing 2: *(Fortsetzung)*

```
 if ( winopen )          /* ist überhaupt ein Fenster geöffnet? */
  {                                                    /* Ja */
   wptr = winptr + winopen - 1;   /* Ptr auf Fensterbeschr. */
   bptr = (VEL *) malloc( BUFLEN( x1 = wptr->x1,
     y1 = wptr->y1, x2 = wptr->x2, y2 = wptr->y2 ) );
   if ( bptr )  /* konnte ein Zwischensp. allokiert werden? */
    {                                                  /* Ja */
     /*-- 1. den Fensterinhalt in Zwischenspeicher holen ---*/
     VioGet( x1, y1, x2, y2, bptr );

     /*-- 2. Bildschirminhalt unter Fenster zurückbringen --*/
     VioPut( x1, y1, x2, y2, aktbuf = wptr->winmem );


     /*-- 3. Cursor versetzen ------------------------------*/
     if ( (x1 <= vspalte ) && (x2 >= vspalte ) &&
          (y1 <= vzeile ) && (y2 >= vzeile ) )
       VioSetCursor( vspalte - x1 + xneu,
                     vzeile - y1 + yneu );


     /*-- 4. Bildschirminhalt an neuer Pos. holen ----------*/
     xdif = xneu - x1;               /* Entfernung: Spalten */
     ydif = yneu - y1;                /* Entfernung: Zeilen */
     VioGet( wptr->x1 = xneu, wptr->y1 = yneu,
             wptr->x2 = x2 = x2+xdif,
             wptr->y2 = y2 = y2+ydif, aktbuf );

     /*-- 5. Fensterinhalt an neue Position bringen --------*/
     VioPut( xneu, yneu, x2, y2, bptr );

     /*-- 6. View-Bereich an die neue Position anpassen ----*/
     viewx1 += xdif;
     viewy1 += ydif;
     viewx2 += xdif;
     viewy2 += ydif;

     free( bptr );       /* Zwischenspeicher wieder freigeben */
     return TRUE;                              /* geschafft! */
    }
   else /* es konnte kein Zwischenspeicher allokiert werden */
    return FALSE;                     /* mit Fehler zurück */
  }
 else
  return FALSE;               /* es ist kein Fenster geöffnet */
}


/***************************************************************
* Funktion          : V i o C o l o r                         *
**-----------------------------------------------------------**
* Aufgabe           : Färbt einen Bildschirmbereich mit       *
*                     einem konstanten Attribut ein.          *
* Eingabe-Parameter: X1, Y1 = obere linke Ecke des Bereichs  *
*                     X2, Y2 = untere rechte Ecke des Ber.    *
*                     FARBE  = Farbe der einzelnen Zeichen    *
* Return-Wert       : keiner                                  *
***************************************************************/

void VioColor(BYTE x1, BYTE y1, BYTE x2, BYTE y2, BYTE farbe)

{
 register VPFAR lptr;                          /* Laufzeiger */
 BYTE i, j;                                /* Schleifenzähler */

 /*-- die einzelnen Zeilen durchlaufen ---------------------*/

 for (j=x2 - x1 + 1 ; y1 <= y2; ++y1)
  {          /* die einzelnen Zeichen der Zeile durchlaufen */
   for (lptr = VPOS(x1, y1), i = j; i ; --i)
    (lptr++)->h.attribut = farbe;        /* die Farbe setzen */
  }
}
```

Listing 2: *(Ende)*

VioGet bedient sich zum Kopieren der Daten aus dem Video-RAM in den angegebenen Puffer des Makros move, das innerhalb des VIO-Moduls definiert wird und je nach dem Speichermodell, unter dem das VIO-Modul kompiliert wird, in einen Aufruf der movedata- oder der memmove-Funktion umgesetzt wird. Diese Unterscheidung ist notwendig, da in den Speichermodellen Small und Medium immer mit NEAR-Pointern gearbeitet wird und dadurch mit Hilfe von memmove keine Daten zwischen dem Puffer innerhalb des Datensegments und dem Video-RAM kopiert werden können. Anstelle von memmove muß hier deshalb auf movedata zurückgegriffen werden.

VioPut

Ein Bildschirmbereich dessen Inhalt zuvor über VioGet in einen Puffer geholt wurde, kann durch den Aufruf von VioPut wieder auf den Bildschirm gebracht werden. Da VioPut den Puffer, aus dem es die Zeichen und ihre Attribute entnimmt, nicht verändert, kann ein Puffer durch mehrmalige Aufrufe von VioPut unter der Angabe unterschiedlicher Bildschirmkoordinaten an verschiedene Bildschirmpositionen gebracht werden.

Auch VioPut bedient sich zum Kopieren der Daten aus dem Puffer in den Video-RAM des move-Makros.

VioWinOpen

Durch den Aufruf dieser Funktion wird ein Bildschirmfenster geöffnet, wobei ein entsprechender Eintrag der Struktur windes angelegt und in ihm die Koordinaten des Fensters, die Koordinaten des aktuellen View-Bereichs sowie die aktuelle Cursorposition abgespeichert werden. Zusätzlich wird der Inhalt des Bildschirmbereichs, den das Fenster überdeckt, mit Hilfe von VioGet in einen Puffer geholt, der über den Heap allokiert wird.

Steht nicht mehr genügend Speicher auf dem Heap bereit, um einerseits den Fenster-Vektor zu vergrößern und andererseits einen Puffer zur Aufnahme des Bildschirmbereichs zu allokieren, liefert die Funktion die Konstante FALSE, die in der Datei VIO.H definiert wird, an den Aufrufer zurück. Konnte das Fenster jedoch einwandfrei geöffnet werden, wird dem Aufrufer die Konstante TRUE übergeben.

VioWinClose

Diese Funktion bildet das Gegenstück zu VioWinOpen und schließt das jeweils zuletzt geöffnete Fenster wieder. Der Aufrufer kann dabei durch Übergabe eines Parameters entscheiden, ob der alte Bildschirminhalt unter dem Fenster wieder hergestellt wird. Spricht sich der Aufrufer dafür aus, wird gleichzeitig auch der Cursor wieder auf die Position gesetzt, die er vor dem Öffnen des Fensters inne hatte. Gleiches gilt für den View-Bereich, der ebenfalls zurückgesetzt wird.

VioSetCursor

Der blinkende Bildschirmcursor wird durch diese Funktion auf die angegebene Bildschirmposition versetzt.

Die aktuelle Cursorposition kann vom Anwendungsprogramm aus jederzeit mit Hilfe der Makros aktz und akts abgefragt werden, die in der Datei VIO.H definiert werden.

Auf der Funktion VioSetCursor basiert auch das Makro VioHideCursor mit dessen Hilfe der Cursor auf eine Position außerhalb des Bildschirm gesetzt und damit unsichtbar gemacht wird.

VioPrint

Der Ausgabe einer Zeichenkette auf dem Bildschirm dient die Funktion VioPrint. Neben der Ausgabeposition auf dem Bildschirm kann der Aufrufer auch die Farbe bzw. das Attribut bestimmen, in der die Zeichen auf dem Bildschirm erscheinen sollen. Ein weiterer Parameter entscheidet darüber, ob der Cursor nach der Ausgabe hinter das letzte ausgegebene Zeichen bewegt wird, oder ob er auf seiner aktuellen Position verharren soll.

Beachten Sie bitte, daß VioPrint alle Zeichen der Zeichenkette unbearbeitet auf den Bildschirm bringt und Steuerzeichen wie z.B. Carriage Return oder Bell als ganz normale ASCII-Zeichen behandelt.

VioPrintf

Diese Funktion ähnelt der printf-Funktion jedoch mit dem Unterschied, daß hier die Ausgabeposition auf dem Bildschirm und die Farbe der auszugebenden Zeichen vom Aufrufer festgelegt wird. Ansonsten können die gewohnten Formatierungskennzeichen von printf (%d, %f etc.) in vollem Umfang benutzt werden.

VioStrep

Immer wieder kommt es vor, daß ein Zeichen mehrmals hintereinander ausgegeben oder in eine Zeichenkette eingebunden werden soll. In einem solchen Fall leistet VioStrep wertvolle Hilfe, indem es einen String generiert, der aus einer bestimmten Anzahl von Zeichen besteht, wobei der Aufrufer sowohl die Anzahl (im Bereich zwischen 1 und 80), als auch das zu wiederholende Zeichen frei definieren kann.

VioFill

Ein bestimmter Bildschirm kann mit Hilfe dieser Funktion mit einem konstanten Zeichen unter einer konstanten Farbe gefüllt werden. Neben den Eckkoordinaten des Bildschirmbereichs muß dieser Funktion auch das Füllzeichen und eine Farbe für dieses Zeichen übergeben werden.

Auf dieser Funktion basieren auch die Makros `VioClear` und `VioClearScreen` aus der Datei VIO.H, mit deren Hilfe der gesamte Bildschirm gelöscht, bzw. ein Teil des Bildschirms mit Leerzeichen gefüllt werden kann.

VioFrame

Fenster werden vom übrigen Bildschirminhalt oft durch eine andere Farbe oder einen Rahmen optisch hervorgehoben. Der letzteren Aufgabe widmet sich die Funktion `VioFrame`, die einen von 4 verschiedenen Rahmen um einen Bildschirmbereich zieht. Der umrahmte Bildschirmbereich und die Farbe des Rahmens können dabei vom Aufrufer frei gewählt werden. Der Rahmentyp muß einer der Konstanten (`EINRA`, `DOPRA` etc.) entsprechen, die in der Datei VIO.H definiert werden.

VioScrollUp

Durch den Aufruf dieser Funktion wird der angegebene Bildschirmbereich um eine oder mehrere Bildschirmzeilen nach oben gescrollt. Sollen die dadurch frei werdenden Bildschirmzeilen am unteren Rand des Bereichs gelöscht werden, kann der Aufrufer ein Farbe für diese Zeilen angeben. Sollen diese Zeilen hingegen nicht gelöscht werden, muß der Aufrufer als Farbe die Konstante `NOCLEAR` aus der Datei VIO.H an `VioScrollUp` übergeben.

VioScrollDown

Diese Funktion entspricht vom Aufruf her der Funktion `VioScrollUp` mit dem Unterschied, daß sie den angegebenen Bildschirmbereich nicht nach oben, sondern nach unten scrollt.

VioSrcollHori

Analog zu den Funktionen `VioScrollUp` und `VioScrollDown` wird ein Bildschirmbereich durch den Aufruf dieser Funktion um eine oder mehrere Spalten nach links oder nach rechts verschoben. Der Aufrufer sollte diese Funktion jedoch nicht direkt, sondern über die beiden Makros `VioScrollLeft` und `VioScrollRight` aus der Include-Datei VIO.H aufrufen.

Auch bei `VioScrollHori` gilt, daß das Löschen der freigewordenen Spalten unterbleibt, wenn als Farbe die Konstante `NOCLEAR` übergeben wird.

VioMoveUp, VioMoveDown, VioMoveLeft, VioMoveRight

Diese Funktionen verschieben jeweils das aktuelle Fenster um die angegebene Anzahl von Spalten oder Zeilen in die entsprechende Richtung. Mit dem Fenster verschiebt sich auch der View-Bereich und die Position des blinkenden Bildschirmcursors, sofern dieser sich innerhalb des zu verschiebenden Fensters befindet. Der Aufrufer hat dabei dafür Sorge zu tragen, daß das Fenster nicht über den Bildschirmrand hinaus verschoben wird.

VioColor

Die letzte Funktion innerhalb des VIO-Moduls dient der Einfärbung eines Bildschirmbereichs mit eine konstanten Farbe bzw. Attribut ohne daß die einzelnen Zeichen innerhalb des Bereichs verändert werden.

Das Demo-Programm

Damit die Beschreibung dieser Funktionen nicht nur graue Theorie bleibt, sondern auch in die Praxis umgesetzt werden kann, demonstriert Ihnen das kleine Demo-Programm (*Listing 3*) den Aufruf fast aller Funktionen des VIO-Moduls. Wie auch das VIO-Modul selbst ist es sehr ausführlich dokumentiert und sollte daher keine Fragen offen lassen. Wie auch das Vio-Modul selbst, kann auch das Demo-Programm in Verbindung mit allen Speichermodellen des Microsoft C-Compilers eingesetzt werden. Um das Demo-Programm beispielsweise unter dem Small-Modell zu kompilieren geben Sie bitte:

```
CL /AS viodemo.c vio.c
```

ein. Ist das VIO-Modul einmal unter dem entsprechenden Speichermodell kompiliert worden, reicht es aus, das Modul mit Ihrem jeweiligen Programm während des Linkvorgangs zu verbinden. Eine erneute Kompilierung des VIO-Moduls muß dann also nicht mehr stattfinden.

Michael Tischer

Michael Tischer *ist Autor mehrerer Fachbücher über DOS und die Programmierung auf Personalcomputer. Sein bekanntestes Buch ist »PC Intern«, das umfassendste deutsche Werk zur PC-Programmierung, das gerade in einer erweiterten und überarbeiteten Auflage herausgekommen ist.*

```
/*************************************************************/
/*                    V I O D E M O . C                      */
/*-----------------------------------------------------------*/
/*    Aufgabe        : Demonstriert die Arbeit mit den ver-  */
/*                     schiedenen Funktionen des VIO-Moduls. */
/*-----------------------------------------------------------*/
/*    Autor          : MICHAEL TISCHER                       */
/*    entwickelt am  :  1.12.1988                            */
/*    letztes Update :  6.12.1988                            */
/*-----------------------------------------------------------*/
/*    Erstellung     : CL /A[S|M|C|L|H] VIODEMO.C VIO.C      */
/*************************************************************/
```

Listing 3: `VIODEMO.C`

```
/*-- Include-Dateien einbinden -------------------------------*/
#include "vio.h"              /* für die Funktionen aus VIO.C */
#include <stdlib.h>                            /* für RAND() */
#include <math.h>                               /* für SIN() */
#include <time.h>                              /* für TIME() */
#include <conio.h>                            /* für GETCH() */

/*-- Konstanten ----------------------------------------------*/
#define FARBE(x)  ( *(farbvek+(x)) )          /* holt Farbe */
#define TEC     0           /* Farbe für "Taste betätigten" */
#define MR      1  /* Farbe des Rahmen im untersten Fenster */
#define MB      2              /* Farbe des ersten Fenster */
#define DFR1    3           /* Rahmenfarbe Demo-Fenster 1 */
#define DFT1    4             /* Textfarbe Demo-Fenster 1 */
#define DFR2    5           /* Rahmenfarbe Demo-Fenster 2 */
#define DFT2    6             /* Textfarbe Demo-Fenster 2 */
#define DFR3    7           /* Rahmenfarbe Demo-Fenster 3 */
#define DFT3    8             /* Textfarbe Demo-Fenster 3 */
#define DFR4    9           /* Rahmenfarbe Demo-Fenster 4 */
#define DFT4   10             /* Textfarbe Demo-Fenster 4 */
#define DFR5   11           /* Rahmenfarbe Demo-Fenster 5 */
#define DFT5   12             /* Textfarbe Demo-Fenster 5 */
#define DFR6   13           /* Rahmenfarbe Demo-Fenster 6 */
#define DFT6   14             /* Textfarbe Demo-Fenster 6 */
#define DFR7   15           /* Rahmenfarbe Demo-Fenster 7 */
#define DFT7   16             /* Textfarbe Demo-Fenster 7 */
#define DFR8   17           /* Rahmenfarbe Demo-Fenster 8 */
#define DFT8   18             /* Textfarbe Demo-Fenster 8 */
#define DFRE   19             /* Rahmenfarbe Ende-Fenster */
#define DFTE   20               /* Textfarbe Ende-Fenster */
#define CRI    21                 /* Farbe für Copyright */

/*-- Typedefs ------------------------------------------------*/
typedef void (*FUP)(void);           /* Pointer auf Funktion */

/*-- Strukturen ----------------------------------------------*/
struct vdata {        /* beschreibt Eingabedaten für VioDemo */
         BYTE rtyp,          /* Rahmentyp für Info-Fenster */
              rcol,            /* Rahmenfarbe als Index */
              tcol;              /* Textfarbe als Index */
         char **vek;        /* Pointer auf Vektor mit Ptr */
         FUP fkt;                  /* Pointer auf Fkt */
        };

/*-- forward Funktions-Deklarationen -------------------------*/
void demo2( void );                     /* Demo-Funktionen */
void demo3( void );
void demo4( void );
void demo5( void );
void demo6( void );
void demo7( void );
void demo8( void );

/*-- globale Variablen ---------------------------------------*/
BYTE colvek[] = {                /* Farben für Farb-Modus */
          BLINK( COL( WEISS, CYAN ) ),      /* TEC */
          COL( GELB, SCHWARZ ),              /* MR */
          COL( GELB, SCHWARZ ),              /* MB */
          COL( WEISS, CYAN ),              /* DFR1 */
          COL( GELB, CYAN ),               /* DFT1 */
          COL( SCHWARZ, VIOLETT ),         /* DFR2 */
          COL( WEISS, VIOLETT ),           /* DFT2 */
          COL( HBLAU, BLAU ),              /* DFR3 */
          COL( WEISS, BLAU ),              /* DFT3 */
          COL( ROT, DGRAU ),               /* DFR4 */
          COL( SCHWARZ, DGRAU ),           /* DFT4 */
          COL( CYAN, BLAU ),               /* DFR5 */
          COL( GELB, BLAU ),               /* DFT5 */
          COL( GELB, ROT ),                /* DFR6 */
          COL( WEISS, ROT ),               /* DFT6 */
          COL( WEISS, CYAN ),              /* DFR7 */
          COL( GELB, CYAN ),               /* DFT7 */
          COL( HBLAU, BLAU ),              /* DFR8 */
          COL( WEISS, BLAU ),              /* DFT8 */
          COL( SCHWARZ, VIOLETT ),         /* DFRE */
          COL( WEISS, VIOLETT ),           /* DFTE */
          COL( GELB, CYAN )                /* CRI */
         },
```

Listing 3: *(Fortsetzung)*

```
     monovek[] = {          /* Attribute für Monochrom-Modus */
               BLINK( INVERS ),                    /* TEC */
               NORMAL,                              /* MR */
               NORMAL,                              /* MB */
               NORMAL,                            /* DFR1 */
               HNORMAL,                           /* DFT1 */
               INVERS,                            /* DFR2 */
               INVERS,                            /* DFT2 */
               NORMAL,                            /* DFR3 */
               HNORMAL,                           /* DFT3 */
               NORMAL,                            /* DFR4 */
               HNORMAL,                           /* DFT4 */
               NORMAL,                            /* DFR5 */
               HNORMAL,                           /* DFT5 */
               NORMAL,                            /* DFR6 */
               HNORMAL,                           /* DFT6 */
               NORMAL,                            /* DFR7 */
               HNORMAL,                           /* DFT7 */
               INVERS,                            /* DFR8 */
               INVERS,                            /* DFT8 */
               NORMAL,                            /* DFRE */
               HNORMAL,                           /* DFTE */
               INVERS                              /* CRI */
              };

BYTE *farbvek;            /* Pointer auf aktiven Farb-Vektor */

/*-- Daten für Demo 1 ----------------------------------------*/
char *t1[] =                      /* Text für Demo-Fenster 1 */
 { "              V I O D E M O              ",
   "",
   "Dieses kleine Demonstrationsprogramm soll Ihnen",
   "die  Leistungsfähigkeit der Funktionen  aus dem",
   "VIO-Modul verdeutlichen.",
   "Zu diesem Zweck werden Ihnen nacheinander  die",
   "wichtigsten  Funktionen  dieses  Moduls  vorge-",
   "stellt...",
   "e"
 };

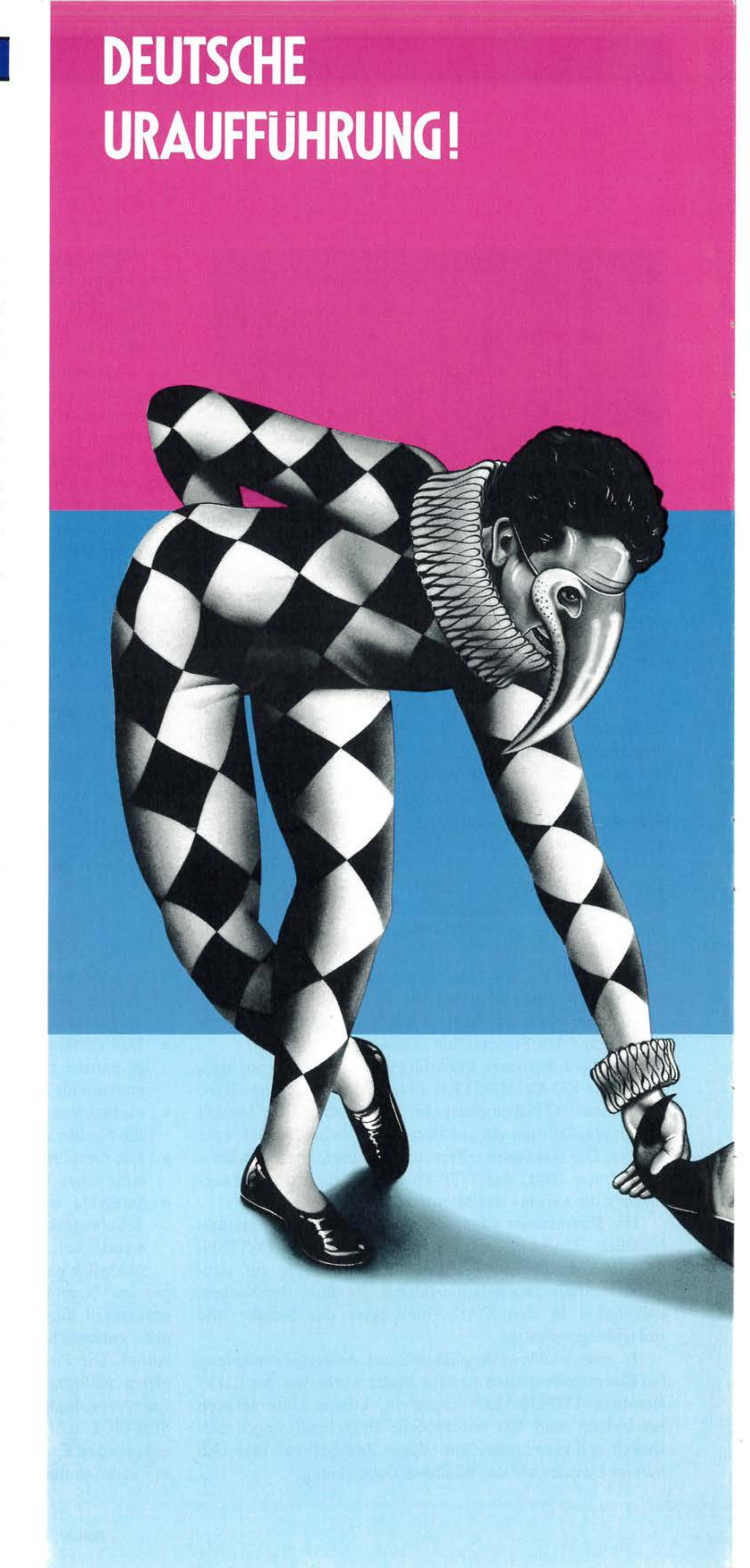
struct vdata df1 = {              /* Daten für Demo-Fenster 1 */
                    EINRA, DFR1, DFT1, t1, (FUP) 0
                   };

/*-- Daten für Demo 2 ----------------------------------------*/
char *t2[] =                      /* Text für Demo-Fenster 2 */
 { "Eingeleitet wird die Arbeit mit den Funktionen des VIO-",
   "Moduls durch den Aufruf der Funktion",
   "",
   "                 VioInit()",
   "",
   "die die internen Variablen des VIO-Moduls initialisiert",
   "und das Modul auf die aktive Videokarte einstellt.",
   "",
   "Danach können Fenster mit Hilfe der Funktionen",
   "",
   "         VioWinOpen() und VioWinClose()",
   "",
   "nach  dem LIFO-Prinzip geöffnet und wieder  geschlossen",
   "werden. Die Anzahl der gleichzeitig geöffneten  Fenster",
   "wird  dabei  nur durch die Größe des  freien  Speicher-",
   "platzes auf dem Heap beschränkt.",
   "e"
 };

struct vdata df2 = {              /* Daten für Demo-Fenster 2 */
                    EINRA, DFR2, DFT2, t2, demo2
                   };

/*-- Daten für Demo 3 ----------------------------------------*/
char *t3[] =                      /* Text für Demo-Fenster 3 */
 { "Der Inhalt eines Bildschirmbereichs kann mit  Hilfe",
   "der Funktion",
   "",
   "                 VioFill()",
   "",
```

Listing 3: *(Fortsetzung)*



»Dynamisch Zeichnen mit Microsoft Excel« war der Titel einer Präsentation am Stand der Management Informations Systeme. Dabei wurde hier nicht eigentlich eine Branchenlösung vorgestellt, sondern ein sinnvolles Zusatzprodukt zu Excel. Die Anwendung »Zeichnen« versetzt Excel in die Lage, einfache geometrische Elemente wie Linien, Polygone, Kreise, Intensitätspfeile und Rechtecke darzustellen. Im Gegensatz zu normalen Zeichenprogrammen aber handelt es sich bei den Zeichen dieser Applikation um »dynamische Zeichen«. Sie werden durch Kalkulationswerte aus Excel-Tabellen definiert. Ändern sich daher die Tabellenwerte, so ändern sich auch automatisch die Zeichen - eine sinnvolle Erweiterung der DTP-Fähigkeiten von Microsoft Excel.

Was der Aussteller zum Zeitpunkt der Frankfurter Windows-Messe noch nicht wußte: Die Anwendung »Zeichnen« wurde von der Jury des PC Magazin-Wettbewerbs um die beste Excel-Anwendung als so gelungen betrachtet, daß sie mit dem ersten Preis bedacht wurde.

Mehr als 400 Händler, Industrievertreter und Journalisten besuchten die Ausstellung deutscher Windows- und Excel-Applikationen. Die Angebotspalette war breit. Sie reichte von grafisch orientierten DTP-Lösungen über ingenieurwissenschaftliche CAD-Anwendungen bis zu Buchhaltungsprogrammen. Christian Wedell, Microsoft-Geschäftsführer und interessiertester Besucher der ganzen Ausstellung, hat versprochen, daß dies nicht die letzte Windows-Schau in deutschen Landen gewesen sein soll. Microsoft will sich im Gegenteil künftig intensiver um die Verbreitung von Produkten unabhängiger, aber mit Microsoft kooperierender Hersteller kümmern. Dies soll im Rahmen eigener, aber auch unabhängiger Messen geschehen. Schließlich hat Microsoft durch MS OS/2 und seine SQL- und Netzwerktechnologie deutlich den Weg in Richtung »Systemhaus« eingeschlagen.

```
int far * far p;
```

p ist ein entfernter Zeiger auf eine far-Integer.

Zum Abschluß

Sie sind nun mit C-Deklarationen und den »Verkehrsregeln« gut genug vertraut, um die Beispiele vom Anfang nachvollziehen zu können. Versuchen Sie diese zu lösen, bevor Sie weitermachen. Um sicher zu sein, daß wir alle übereinstimmen:

```
struct vtag far *
(far * const far var[5])();
```

bedeutet, daß var ein konstantes entferntes Array von 5 Zeigern auf far-Funktionen ist, die Zeiger auf die far-Strukturen vtag liefern.





Es geht Microsoft nicht mehr nur darum, gute Standardsoftware zu produzieren. Als einziger Komplettanbieter von Betriebssystemen, Netzwerklösungen, Entwicklungswerkzeugen und Standardsoftware wird Microsoft in Zukunft als Systemhaus Anwendern bei der Definition und Entwicklung von Lösungen zur Seite stehen. MS OS/2 als herstellerunabhängiges Softwarekonzept macht es nötig, daß unabhängige Softwarehäuser Systemberatung leisten - unabhängig von der eingesetzten Hardware. In diese Rolle will Microsoft hineinwachsen.

Die Vorstellung von System- und Branchenlösungen mit Windows- und Excelprodukten war auf diesem Weg ein Anfang. Für den Handel ist es wichtig zu vermerken, daß dieser Anfang gemeinsam mit ihm gesucht wird. Denn beim Händler liegt entscheidendes Knowhow für die Systemberatung des Endkunden. »Von der Kooperation technologisch versierter Händler mit Microsoft hängt unser beider Erfolg ab.« So lautete denn auch die Schlußfolgerung von Dr. Jochen Haink, Gesamtvertriebsleiter bei Microsoft, vor den autorisierten Microsoft-Händlern und -Distributoren.

Michael Bülow

Es ist wirklich nicht schwer, oder? Lassen Sie uns sehen, ob Sie es wirklich verstanden haben. Was wird mit dem folgenden deklariert?

```
unsigned long (far * (far * const (far *
far const v[2])[4])())[6];
```

Als letzte Anregung können Sie sich überlegen, ein Programm zu schreiben, das als Eingabe eine C-Deklaration oder eine deutsche Umschreibung erhält, und entweder eine Umschreibung oder die entsprechende C-Deklaration liefert. Beachten Sie auch die Fälle, bei denen es mehrere Deklaratoren gibt, wie zum Beispiel

```
int far *p, far q;
```

Sollte das passieren, dann brechen Sie am besten ab und geben eine geharnischte Warnung aus, die erklärt, daß es sich dabei um schlechten Programmierstil handelt. Viel Vergnügen!

Greg Comeau

```
    "mit einem konstanten Zeichen und einer  konstanten",
    "Farbe gefüllt werden. Neben dem ASCII-Code des Zei-",
    "chens und der Farbe müssen dabei natürlich auch die",
    "die  Koordinaten der oberen linken und der  unteren",
    "rechten Ecke des angesprochenen  Bildschirmbereichs",
    "angegeben werden.",
    "Wie auch bei allen anderen Funktionen,  die die An-",
    "gabe  eines Bildschirmbereichs  erwarten, kann  man",
    "sich dabei  mit Hilfe  der Makros VL(), VO(),  VR()",
    "und VU() auf die Koordinaten des aktuellen Fensters",
    "beziehen.",
    "e"
 };

struct vdata df3 = {                 /* Daten für Demo-Fenster 3 */
                    EINRA, DFR3, DFT3, t3, demo3
                   };
/*-- Daten für Demo 4 ---------------------------------------*/
char *t4[] =                         /* Text für Demo-Fenster 4 */
 { "Ein beliebiger Bildschirmbereich kann mit Hilfe",
    "der Funktion",
    "",
    "                  VioColor()",
    "",
    "eingefärbt werden, ohne daß dadurch die Zeichen",
    "innerhalb des Bereichs beeinflußt werden.",
    "e"
 };

struct vdata df4 = {                 /* Daten für Demo-Fenster 4 */
                    EINRA, DFR4, DFT4, t4, demo4
                   };

/*-- Daten für Demo 5 ---------------------------------------*/
char *t5[] =                         /* Text für Demo-Fenster 5 */
 { "Um einen beliebigen Bilschirmbereich kann mit Hilfe",
    "der Funktion",
    "",
    "                  VioFrame()",
    "",
    "einer von vier verschiedenen Rahmen gezogen werden.",
    "e"
 };

struct vdata df5 = {                 /* Daten für Demo-Fenster 5 */
                    EINRA, DFR5, DFT5, t5, demo5
                   };

/*-- Daten für Demo 6 ---------------------------------------*/
char *t6[] =                         /* Text für Demo-Fenster 6 */
 { "Zeichenketten können ab einer bestimmten  Bildschirm-",
    "position mit Hilfe der Funktionen",
    "",
    "              VioPrint() und VioPrintf",
    "",
    "ausgegeben werden. Während VioPrint() einen einfachen",
    "ASCII-String  auf dem  Bildschirm  ausgibt,  kann der",
    "Funktion VioPrintf analog  zur Funktion  printf() ein",
    "Formatstring  und die dazugehörigen  Argumente  über-",
    "geben werden.",
    "e"
 };

struct vdata df6 = {                 /* Daten für Demo-Fenster 6 */
                    EINRA, DFR6, DFT6, t6, demo6
                   };

/*-- Daten für Demo 7 ---------------------------------------*/
char *t7[] =                         /* Text für Demo-Fenster 7 */
 { "Der Inhalt eines Bildschirmbereichs kann mit Hilfe",
    "der Funktionen",
    "",
    "        VioScrollLeft(), VioScrollRight(),",
    "        VioScrollUp() und VioScrollDown()",
    "",
    "in alle vier Himmelsrichtungen gescrollt werden.",
    "e"
 };
```

Listing 3: *(Fortsetzung)*

```
struct vdata df7 = {                 /* Daten für Demo-Fenster 7 */
                    EINRA, DFR7, DFT7, t7, demo7
                   };

/*-- Daten für Demo 8 ---------------------------------------*/
char *t8[] =                         /* Text für Demo-Fenster 7 */
 { "Das jeweils aktuelle Fenster kann mit Hilfe der",
    "Funktionen",
    "",
    "          VioMoveLeft(), VioMoveRight(),",
    "          VioMoveUp() und VioMoveDown()",
    "",
    "über  den  Bildschirm geschoben  und durch  den",
    "Aufruf der Funktion",
    "",
    "                  VioSetWin()",
    "",
    "an  eine beliebige Bildschirmposition  gebracht",
    "werden.",
    "e"
 };
struct vdata df8 = {                 /* Daten für Demo-Fenster 8 */
                    EINRA, DFR8, DFT8, t8, demo8
                   };

/*-- Daten für Verabschiedung -------------------------------*/
char *te[] =                         /* Text für Verabschiedung */
 { "Das war's",
    "",
    "Auf wiederstarten ...",
    "e"
 };

struct vdata dfe = {                 /* Daten für Demo-Fenster 1 */
                    EINRA, DFRE, DFTE, te, (FUP) 0
                   };

struct vdata *demovek[] = { /* Vektor mit Demo-Beschreibern */
                    &df1, &df2, &df3, &df4, &df5,
                    &df6, &df7, &df8, &dfe
                   };

/************************************************************
* T A S T E                                                 *
************************************************************/

void taste( void )
{
 #define TEND  ( VioGetCols() - 5 ) /* Endspalte des Textes */
 #define SPOS  ( TEND - sizeof tb )        /* Startposition */

 static char tb[] = " bitte Taste betätigen ";

 VioPrint( SPOS, 0, FARBE( MR ), FALSE, "┤" );
 VioPrint( SPOS+1, 0, FARBE( TEC ), FALSE, tb );
 VioPrint( TEND, 0, FARBE( MR ), FALSE, "├" );
 getch();                             /* auf Taste warten */
 VioPrint( SPOS, 0, FARBE( MR ), FALSE,
          VioStrep( '─', sizeof tb + 1 ) );

 #undef TEND                   /* Konstanten wieder löschen */
 #undef SPOS
}

/************************************************************
* D E M O 2 (Fenster öffnen und wieder schließen)           *
************************************************************/

void demo2( void )
{
 #define MAX WIN 200          /* amximale Anzahl von Fenstern */
 #define ZUFALL(x) ( rand() % ((x)+1) )

 BYTE i = 0,                              /* Schleifenzähler */
      weiter,             /* TRUE, wenn Fenster geöffnet wurde */
      sx, sy,                   /* Startposition des Fenster */
      ex, ey,                    /* Endposition des Fenster */
      col;             /* Farbe für Rahmen und Fensterinhalt */
```

Listing 3: *(Fortsetzung)*

```
 srand( (int) time( (time_t *) 0 ) );  /* Zufallsgen. init. */
 do                                          /* Aufbauschleife */
  {
   sx = 3 + ZUFALL( VioGetCols()-30 );           /* Startspalte */
   sy = 3 + ZUFALL( VioGetLines()-10 );           /* Startzeile */
   ex = sx + 10 + ZUFALL( VioGetCols()- 15 - sx );
   ey = sy + 3 + ZUFALL( VioGetLines() - 6 - sy );
   if ( weiter = VioWinOpen( sx, sy, ex, ey ) )
    {                     /* Fenster konnte geöffnet werden */
     if ( VioIsColor() )          /* Farb-Darstellung möglich? */
      col = COL( ZUFALL( 15 ), ZUFALL( 7 ) );            /* Ja */
     else
      col = ( ZUFALL( 1 ) ) ? NORMAL : INVERS;
     VioFrame( VL(0), VO(0), VR(0), VU(0), EINRA, col );
     VioFill( VL(1), VO(1), VR(-1), VU(-1),
              ZUFALL(25) + 'A', col );
     VioPrintf(VR(-8), VO(0), col, TRUE, "┤ %3d ├", i+1);
    }
  }
 while (weiter && ++i<MAX_WIN);
 taste();                                  /* auf Taste warten */
 for ( ; i--; )              /* alle Fenster wieder schließen */
  VioWinClose( TRUE );

 #undef MAX_WIN                   /* Konstanten wieder löschen */
 #undef ZUFALL
}

/****************************************************************
* D E M O 3 (Fenster immer wieder mit Zeichen füllen)          *
****************************************************************/

void demo3( void )
{
 BYTE i;                                    /* Schleifenzähler */

 VioHideCursor();                  /* Cursor unsichtbar machen */
 VioWinOpen( 20, 6, 78, 23 );                 /* Fenster öffnen */
 VioFrame( VL(0), VO(0), VR(0), VU(0), EINRA, FARBE( DFR1 ) );
 for ( i=' ' ; i < ']'; ++i)
  VioFill( VL(1), VO(1), VR(-1), VU(-1), i, FARBE( DFR1 ) );
 taste();                                  /* auf Taste warten */
 VioWinClose( TRUE );              /* Fenster wieder schließen */
}

/****************************************************************
* D E M O 4 (Fenster in den verschiedensten Farben einf.)       *
****************************************************************/

void demo4( void )
{
 int i;                                     /* Schleifenzähler */

 VioHideCursor();                  /* Cursor unsichtbar machen */
 VioWinOpen( 40, 8, 78, 20 );                 /* Fenster öffnen */
 VioFrame( VL(0), VO(0), VR(0), VU(0), EINRA, FARBE( DFR2 ) );
 VioFill( VL(1), VO(1), VR(-1), VU(-1), '▪', FARBE( DFR2 ) );
 if ( VioIsColor() )              /* Farb-Darstellung möglich? */
  for (i=0; i<256; )             /* JA, Farben durchlaufen */
   VioColor( VL(1), VO(1), VR(-1), VU(-1), i++ );
 else
  for (i=0; i<10; i++)
   {
    VioColor( VL(1), VO(1), VR(-1), VU(-1), NORMAL );
    VioColor( VL(1), VO(1), VR(-1), VU(-1), UNDERLINE );
    VioColor( VL(1), VO(1), VR(-1), VU(-1), HNORMAL );
    VioColor( VL(1), VO(1), VR(-1), VU(-1), INVERS );
   }
 taste();                                  /* auf Taste warten */
 VioWinClose( TRUE );              /* Fenster wieder schließen */
}

/****************************************************************
* D E M O 5 (unterschiedliche Rahmentypen darstellen)           *
****************************************************************/
void demo5( void )
{
 BYTE i;
```

Listing 3: *(Fortsetzung)*

```
 VioHideCursor();                  /* Cursor unsichtbar machen */
 VioWinOpen( 1, 1, ( VioGetCols() >> 1 ) - 1 ,
             ( VioGetLines() >> 1 ) -1 );
 VioFrame( VL(0), VO(0), VR(0), VU(0), EINRA, FARBE( DFR1 ) );
 VioClear( VL(1), VO(1), VR(-1), VU(-1), FARBE( DFR1 ) );

 VioWinOpen( VioGetCols() >> 1, 1, VioGetCols()-2,
             ( VioGetLines() >> 1 ) - 1);
 VioFrame( VL(0), VO(0), VR(0), VU(0), DOPRA, FARBE( DFR2 ) );
 VioClear( VL(1), VO(1), VR(-1), VU(-1), FARBE( DFR2 ) );

 VioWinOpen( 1, VioGetLines() >> 1, ( VioGetCols() >> 1 ) - 1,
             VioGetLines() - 2 );
 VioFrame(VL(0), VO(0), VR(0), VU(0), VOLLRA, FARBE( DFR3 ));
 VioClear( VL(1), VO(1), VR(-1), VU(-1), FARBE( DFR3 ) );

 VioWinOpen( VioGetCols() >> 1, VioGetLines() >> 1,
             VioGetCols() - 2, VioGetLines() - 2);
 VioFrame(VL(0), VO(0), VR(0), VU(0), PUNKTRA, FARBE( DFR4 ));
 VioClear( VL(1), VO(1), VR(-1), VU(-1), FARBE( DFR4 ) );

 taste();                                  /* auf Taste warten */
 for ( i=0; i<4; ++i )
  VioWinClose( TRUE );             /* Fenster wieder schließen */
}

/****************************************************************
* D E M O 6 (Bildschirmausgaben über VioPrintf)                 *
****************************************************************/

void demo6( void )
{
 #define PI 3.141592654

 BYTE w,                                              /* Winkel */
      i;                                    /* Schleifenzähler */

 VioWinOpen( 20, 9, 70, 21 );                 /* Fenster öffnen */
 VioFrame( VL(0), VO(0), VR(0), VU(0), DOPRA, FARBE( DFR1 ) );
 VioClear( VL(1), VO(1), VR(-1), VU(-1), FARBE( DFT1 ) );

 VioPrint( VL(2), VO(2), FARBE( DFT2 ), FALSE,
           "Ausgaben über VioPrintf()" );
 for (i=w=0; i<7; ++i, w+=15)
  VioPrintf( VL(4), VO(4+i), FARBE( DFT1 ), TRUE,
            "Winkel: %2d°, Bogenmaß: %6.4f, SIN()=%6.4f", w,
            (float) w*PI/180.0, sin( (float) w*PI/180.0 ));
 taste();                                  /* auf Taste warten */
 VioWinClose( TRUE );              /* Fenster wieder schließen */
}

/****************************************************************
* D E M O 7 (Inhalt eines Fensters in versch. Richt. scr.)      *
****************************************************************/

void demo7( void )
{
 static char str[] = "\x00";
 BYTE i, j, c;                              /* Schleifenzähler */

 VioHideCursor();                  /* Cursor unsichtbar machen */
 VioWinOpen( 2, 2, 77, 21 );                  /* Fenster öffnen */
 VioFrame(VL(0), VO(0), VR(0), VU(0), VOLLRA, FARBE( DFR7 ));
 for ( i=c=1; i<19; ++i )           /* die Zeilen durchlaufen */
  for ( j=1; j<75; ++j, ++str[0] )   /* die Spalten durchl. */
   {
    if ( ++str[0] == '\0') /* Zeichen inkrementieren */
     ++str[0]; /* NUL-Zeichen nicht ausgeben */
    VioPrint( VL(j), VO(i), FARBE( DFT7 ), TRUE, str );
   }
 for ( i=0; i<7 ; ++i )
  VioScrollUp(VL(1), VO(2), VR(-1), VU(-1), 1, FARBE( DFT7 ));
 taste();                                  /* auf Taste warten */
 for ( i=0; i<7 ; ++i )
  VioScrollDown(VL(1),VO(1),VR(-1),VU(-2),1,FARBE( DFT7 ));
 taste();                                  /* auf Taste warten */
 for ( i=0; i<15 ; ++i )
  VioScrollLeft(VL(2),VO(1),VR(-1),VU(-1),1,FARBE( DFT7 ));
```

Listing 3: *(Fortsetzung)*

```
 taste();                                        /* auf Taste warten */
 for ( i=0; i<15 ; ++i )
  VioScrollRight(VL(1),VO(1),VR(-2),VU(-1),1,FARBE( DFT7 ));
 taste();                                        /* auf Taste warten */
 VioWinClose( TRUE );                    /* Fenster wieder schließen */
}

/***************************************************************
* D E M O 8 (Fenster über den Bildschirm verschieben)          *
***************************************************************/

void demo8( void )
{
 BYTE links,                                          /* Stop links */
      rechts,                                        /* Stop rechts */
      oben,                                            /* Stop oben */
      unten,                                          /* Stop unten */
      xstep;                     /* Grenzverschiebung in X-Richtung */

 links = oben = 0;                               /* Grenzen setzen */
 rechts = VioGetCols() - 1;
 unten = VioGetLines() - 1;
 xstep = rechts / unten;                /*Grenzverschiebung setzen */
 VioWinOpen( 0, 0, 10, 5 );                      /* Fenster öffnen */
 VioFrame( VL(0), VO(0), VR(0), VU(0), DOPRA, FARBE( DFR8 ) );
 VioSetCursor( VL(1), VO(1) );
 VioFill(VL(1), VO(1), VR(-1), VU(-1), '\x01',FARBE( DFT8 ));
 while ( VU(0)<unten && VR(0)<rechts )
  {
   while ( VU(0) != unten )            /* bereits unten angelangt? */
    VioMoveDown( 1 );                                      /* Nein */
   --unten;                          /* Untergrenze dekrementieren */
   while ( VR(0) != rechts )          /* bereits rechts angelangt? */
    VioMoveRight( 1 );                                     /* Nein */
   rechts -= xstep;               /* rechte Grenze dekrementieren */
   while ( VO(0) != oben )              /* bereits oben angelangt? */
    VioMoveUp( 1 );                                        /* Nein */
   ++oben;                            /* Obergrenze inkrementieren */
   while ( VL(0) != links )            /* bereits links angelangt? */
    VioMoveLeft( 1 );                                      /* Nein */
   links += xstep;                 /* linke Grenze inkrementieren */
  }
 taste();                                        /* auf Taste warten */
 VioSetWin( 0, 0 );               /* Fenster in obere linke Ecke */
 taste();                                        /* auf Taste warten */
 VioSetWin( 0, VioGetLines()-6 );             /* untere linke Ecke */
 taste();                                        /* auf Taste warten */
 VioSetWin( VioGetCols()-11, VioGetLines()-6 ); /* unten r. */
 taste();                                        /* auf Taste warten */
 VioSetWin( VioGetCols()-11, 1 );                  /* oben rechts */
 taste();                                        /* auf Taste warten */
 VioWinClose( TRUE );                    /* Fenster wieder schließen */
}

/***************************************************************
* V I O D E M O                                                *
***************************************************************/
```

Listing 3: *(Fortsetzung)*

```
void viodemo( struct vdata * vptr)
{
 BYTE i, j, k,                                 /* Schleifenzähler */
      wcol,                    /* linke Spalte des Textfensters */
      wrow;                      /* ober Zeile des Textfensters */

 char **cptr;   /* zum Durchlaufen der einzelnen Textzeilen */

 /*-- längste Textzeile ermitteln--------------------------- */

 for (i=j=0, cptr=vptr->vek; strcmp(*cptr, "e"); ++i, ++cptr)
  if ( (k=strlen(*cptr)) > j)     /* bisher längster String? */
   j = k;                                    /* Ja, Länge merken */

 wcol = ( VioGetCols() - j - 4 ) >> 1;
 wrow = ( VioGetLines() - i - 3 ) >> 1;
 VioWinOpen(wcol, wrow, wcol+j+3, wrow+i+3);
 VioClear( VL(0), VO(0), VR(0), VU(0), k = FARBE(vptr->tcol));
 if ( vptr->rtyp != KEINRA )                    /* Rahmen ziehen? */
  VioFrame( VL(0), VO(0), VR(0), VU(0),                  /* Ja */
            vptr->rtyp, FARBE(vptr->rcol));

 for (j=2, cptr=vptr->vek; i; --i, ++cptr, ++j)
  VioPrint( VL(2), VO(j), k, TRUE, *cptr);
 taste();
 if ( vptr->fkt != (FUP) 0 )                   /* Fkt. aufrufen? */
  ( vptr->fkt )();                                         /* Ja */
 VioWinClose( TRUE );           /* das Fenster wieder schließen */
}


/***************************************************************
*                    H A U P T P R O G R A M M                 *
***************************************************************/

main()
{
 static char crstring[] =
   " VIODEMO - (c) 1988 by Michael Tischer ";

 BYTE i;                                      /* Schleifenzähler */

 VioInit();                         /* Vio-Modul initialisieren */
 farbvek = VioIsColor() ? colvek : monovek; /* Farben laden */

 /*-- gesamten Bildschirm als Fenster öffnen ---------------*/
 VioWinOpen( 0, 0, VioGetCols()-1, VioGetLines()-1 );
 VioClear( VL(1), VO(1), VR(-1), VU(-1), FARBE( MB ) );
 VioFrame( VL(0), VO(0), VR(0), VU(0), DOPRA, FARBE( MR ) );
 VioPrint( VL((VCOL - sizeof crstring) >> 1), VU(0),
           FARBE( CRI ), FALSE, crstring );

 /*-- die einzelnen Demos ausführen-------------------------*/
 for (i=0; i< (sizeof demovek / sizeof(struct vdata *)); ++i)
  viodemo( demovek[ i ] );

 VioWinClose( TRUE );        /* Haupt-Fenster wieder schließen */
}
```

Listing 3: *(Ende)*

Das Verständnis von Deklarationen ist die Basis für das Verständnis von C:

Komplexe C-Deklarationen verständlich gemacht

Die Sprache C gibt es jetzt schon seit etlichen Jahren, aber viele Teile ihrer Syntax werden sogar von guten Programmierern noch immer nicht verstanden. Der Grund dafür ist das Fehlen einer einheitlichen Dokumentation, die alle Möglichkeiten, im besonderen auch Zeiger, behandelt, die in der Sprache verfügbar sind, einschließlich der Erweiterungen des American National Standards Instituts und von Microsoft.

Solange diese Neuerungen nicht klar oder sogar falsch verstanden werden, nützen Programmierer die Sprache C unglücklicherweise nicht voll aus. Statt dessen werden (sogar von sehr guten Programmierern) Programme geschrieben, die unnötig kompliziert und manchmal sogar nicht ganz korrekt sind. Der Zweck dieses Artikels ist es, einige typische Eigenschaften von C-Deklarationen, die sowohl Anfänger als auch Experten verwirren, klarzustellen. Sehen wir uns am Anfang eine Deklaration an, die viele C-Programmierer für schwerverständlich halten:

```
struct vtag far * (far * const far var [5])();
```

Wenn Sie auf Anhieb verstehen, was diese Deklaration bedeutet, springen Sie gleich zum Ende des Artikels und bearbeiten die Deklarationen am Ende zur Übung. Wenn Sie soweit sind, wie ich noch vor kurzer Zeit war, das heißt, Sie denken, daß Sie es verstehen, sind sich aber nicht sicher, dann sollten Sie weitermachen. Sie werden in der Lage sein, C-Deklarationen wie die vorstehende zu lesen und zu gebrauchen, Sie werden sogar das Lesen und Schreiben von C-Deklarationen anderen erklären können. Nur dann können Sie die volle Leistungsfähigkeit der Sprache voll nutzen.

Deklarationssyntax

Um eine Sprache zu benutzen, müssen Sie etwas über die Struktur und Syntax wissen. Zuerst müssen Sie beim Lesen einer C-Deklaration feststellen wie sie organisiert ist. Innerhalb der vorgegebenen Anordnung einer Deklaration können verschiedene Attribute angegeben werden. Abhängig von den verwendeten Attributen kann der Typ des Bezeichners festgestellt werden. Die Syntax für die explizite Deklaration von Bezeichnern in C ist in *Abbildung 1* dargestellt.

Eine Deklaration kann viele Angaben aus Abbildung 1 enthalten, aber sie muß mindestens einen Typ und einen Deklarator enthalten. Beachten Sie, daß einige Compiler die Angabe eines Bezeichners nur außerhalb einer Funktion akzeptieren; C-Programmierer sollten diese Codierpraxis meiden, da es schlechter Programmierstil ist, zu Fehlern führen kann und bei der Wartung Probleme auftreten können. Darüber hinaus ist diese Praxis durch die bevorstehende ANSI-Normung veraltet.

Die Syntax einer C-Deklaration lautet:

```
Speicherklasse Typ Qualifizierer Deklarator = Initialisierung;
```

Wobei *Speicherklasse* folgendes sein kann:

```
typedef
extern
static
auto
register
```

Typ kann eines oder mehrere dieser Schlüsselwörter sein:

```
void
char
short, int, long
float, double
signed, unsigned
struct ...
union ...
typedef type
```

Ein *Deklarator* enthält einen Bezeichner und einen oder mehrere, oder auch keine der folgenden Zeichen in einer Vielzahl von Kombinationen:

```
*
()
```

Eventuell sind diese geklammert, um unterschiedliche Bindungen auszudrücken.

Abbildung 1: *Die Standardsyntax für C-Deklarationen*

Deklarationen: Theorie

Viele von uns können Deklarationen wie die folgenden lesen:

```
int     i;
char    *p;
```

Dank Brian W. Kernighan und Dennis M. Ritchie und ihrem Buch »The C Programming Language« (Prentice-Hall, Inc. 1978) - oder K&R, wie ich es von jetzt an nenne - können wir sogar noch folgende verstehen:

```
int     *ia[3];
int     (*ia)[3];
```

Bisher haben wir uns dies einfach gemerkt. Zu unserem Glück deckt dies 85% der vorkommenden Deklarationen ab. Das Verständnis der restlichen 15% bereitet jedoch größere Schwierigkeiten.

1. Klammern Sie Deklarationen so, als ob es Ausdrücke wären.
2. Suchen Sie die innerste Klammer.
3. Sagen Sie »Bezeichner ist«, wobei Bezeichner der Name der Variablen ist.
   Sagen Sie »ein Array von X« wenn Sie [X] sehen.
   Sagen Sie »Zeiger auf« wenn Sie * sehen.
4. Gehen Sie zur nächsten Klammerebene.
5. Wenn es weiter geht, machen Sie bei 3 weiter.
6. Sonst sagen Sie »Typ« für den verbleibenden Typ auf der linken Seite (wie zum Beispiel `short int`).

Abbildung 2: *Regeln zum Lesen und Schreiben von K&R-Deklarationen*

Vielen Programmieren fehlt das Verständnis für komplexe Deklarationen und die Benutzung der damit deklarierten Bezeichner. Wegen schlechter Dokumentationen ist für gewöhnlich Raten der einzige Ausweg. Über kurz oder lang werden Sie auf den Bauch fallen, da das Raten zu Verallgemeinerungen führt, die nicht notwendigerweise richtig sind. Auch wenn Sie dem Sinn sehr nahe kommen, kann der vom Compiler erzeugte Code unter Umständen stark vom Gewünschten abweichen.

Ich wünschte, das Rätselraten wäre mir erspart geblieben, als ich C lernte. Das ist um so trauriger, da die Theorie von Deklarationen sehr einfach ist. Sie brauchen nur wissen, daß Deklarationen auf der Hierarchie der C-Operatoren beruhen, derselben, die Sie auch beim Erstellen von Ausdrücken benutzen. Im Fall der Deklarationen bedeutet dies für die Auswertungsreihenfolge:

( ) oder [ ]	haben höchste Priorität, Auswertung von links nach rechts
*	hat niedrigste Priorität

wobei Klammerung die normalen Vorrangregeln ändert. Das ist schon alles. Mit diesem Wissen brauchen wir nur noch Regeln zu formulieren, so wie sie in *Abbildung 2* vorgeschlagen werden.

Lassen Sie uns einige Beispieldeklarationen durchmachen. Benutzen wir die Seite 200 von K&R als Referenz, um die in *Abbildung 3* dargestellten Deklarationen zu studieren. Diese sind entsprechend der Vorrangregel-Tabelle der Sprache C geklammert.

Wenn eine Deklaration einmal geklammert ist, besteht die Entzifferung lediglich darin, zu sagen, was jeder geklammerte Ausdruck bedeutet. Dies gleicht der Klammerung von arithmetischen Ausdrücken, bei denen eine bestimmte Addition Vorrang vor einer bestimmten Multiplikation haben kann. Die einzige Schwierigkeit besteht darin, daß Multiplikation und Division binäre Operatoren sind (sie arbeiten mit zwei Operanden), wogegen wir es mit unären Operatoren zu tun haben, die nur einen Operanden benötigen.

```
int     i;
int     (i);
|        |
|        v
|       i ist
v
eine Integer
```

```
int     *i;
int     (*(i));
|        | |
|        | v
|        | i ist
|        v
|       ein Zeiger auf
v
eine Integer
```

```
int     *i[3];
int     (*((i)[3]));
|        | |
|        | v
|        | i ist ein Array von 3
|        v
|       Zeigern auf
v
Integer
```

```
int     (*i)[3];
int     ((*(i))[3]);
|         | |
|         | v
|         | i ist
|         v
|        ein Zeiger auf ein Array von 3
v
Integern
```

```
int     *i();
int     (*((i)()));
|        | |
|        | v
|        | i ist eine Funktion mit dem Rückgabewert
|        v
|       Zeiger auf
v
Integer
```

```
int     (*i)();
int     ((*(i))());
|         | |
|         | v
|         | i ist
|         v
|        ein Zeiger auf eine Funktion mit dem Rückgabewert
v
Integer
```

Abbildung 3: *Die Interpretation von Deklarationen durch Klammerung*

Beachten Sie, daß jede geklammerte Deklaration in Abbildung 3 auch syntaktisch eine gültige Deklaration ist. Sie werden nicht nur wegen ihrer gleichen Bedeutung gezeigt.

1. Gegeben: Nicht terminierende Attribute sind [], ( ) und *, also Arrays, Funktionen, und Zeiger.

2. Beachten Sie die Rechts-nach-links-Regel: Schauen Sie nach rechts (innerhalb der Klammern), nehmen Sie das Attribut, falls vorhanden, schauen Sie dann nach links, und nehmen, falls vorhanden, dieses Attribut.

3. Das Übersetzen einer C-Deklaration ins Deutsche:
   a. Stellen Sie den Bezeichner der Deklaration fest. Sagen Sie »Bezeichner ist« wobei der Bezeichner der Name der Variablen ist.
   b. Schauen Sie rechts vom Bezeichner nach den Attributen ( ) oder [ ]. Bedenken Sie: Es kann auch keines vorhanden sein.
   Sagen Sie »ein Array von« wenn Sie [] sehen.
   Sagen Sie »ein Array von x« für alle [x], die Sie sehen.
   Sagen Sie »ein x-mal-y Array von« wenn Sie [x][y] sehen.
   Sagen Sie »ein x-mal-y mal ... Array von« wenn Sie [x][y][...] sehen.
   Sagen Sie »Funktion mit Rückgabewert« für ( ), wenn das zuletzt gefundene rechte Attribut [] war.
   Sagen Sie »eine Funktion mit Rückgabewert« für ( ).
   c. Schauen Sie jetzt auf die linke Seite des Bezeichners (wegen der Rechts-nach-links-Regel), und schauen Sie nach weiteren Attributen. Es kümmern uns hier nur Sterne; alles andere wäre ein Fehler. Denken Sie daran, daß hier auch nichts stehen kann. Beachten Sie auch die Klammern. Für jedes * sagen Sie »Zeiger auf«, wenn das zuletzt rechts gefundene Attribut [] war und das jetzige Attribut * ist, andernfalls sagen Sie »ein Zeiger auf«, wenn Sie * sehen.
   d. Schauen Sie wieder rechts nach Attributen. Hier können auch keine sein. Seien Sie hier vorsichtig mit Klammern. Falls dort welche stehen, gehen Sie zurück nach b.
   e. Es sollte ein terminierendes Attribut wie eines der folgenden übrigbleiben: `char`, `int`, `short`, `long`, `float`, `double`, `struct`, `union`, und/oder einer ihrer Modifizierer `signed`, `unsigned`, `static`, `register`, und `extern`.
   Sagen Sie »Struktur vom Typ y« für `struct y`.
   Sagen Sie »Union vom Typ y« für `union y`.
   Sagen Sie »Attribut« und lesen Sie von links nach rechts (wörtlich) die terminierenden Attribute.

4. Regeln zum Umsetzen vom Deutschen in C-Deklarationen:
   a. Schreiben Sie »Bezeichner«, wobei Bezeichner der Name der Variablen ist.
   b. Wir müssen uns mit einem Flag merken, ob das letzte Attribut bei der Verarbeitung ein Stern war. Wir wollen unser Flag »Aktiver-*« nennen und es zu Beginn auf 0 setzen.
   c. Schreiben Sie * zur linken dessen was dasteht, solange Sie in der Beschreibung »Zeiger auf« sehen. Setzen Sie Aktiver-* = 1 (merken, daß das letzte Attribut ein Stern war).
   d. Schreiben Sie `(bisher_aufgeschriebene_Attribute )` wenn Aktiver-* = 1.
   Schreiben Sie [x] rechts, wenn Sie »Array von x« sehen.
   Schreiben Sie [x][y]... rechts, wenn Sie »ein x-mal-y ... Array von« sehen.
   Schreiben Sie [] rechts, wenn Sie »Array von« sehen.
   Schreiben Sie ( ) rechts, wenn Sie »Funktion mit Rückgabewert« sehen.
   e. Machen Sie mit b weiter, wenn noch weitere nicht terminierende Attribute vorhanden sind.
   f. Schreiben Sie `einfaches Attribut` links von allem, wobei das einfache Attribut eines oder eine Kombination der terminierenden Schlüsselwörter ist, die in Abbildung 1 gezeigt wurden.
   Schreiben Sie ; rechts von allem.

5. Anmerkungen:
   a. Sie können kein Array von Funktionen haben. Sie können aber ein Array von Zeigern auf Funktionen haben. Die Deklaration `int a[5]( )` ist ungültig.
   b. Eine Funktion kann kein Array zurückliefern. Eine Funktion kann aber einen Zeiger auf ein Array liefern. Die Deklaration `int a( )[]` ist ungültig.
   c. Eine Funktion kann keine andere Funktion als Ergebnis liefern, sondern nur einen Zeiger darauf, was bedeutet, daß `int a( )( )` ungültig ist.

Abbildung 4: *Lesen und Schreiben von K&R-Deklarationen*

Deklaration: Anwendung

Obwohl die in Abbildung 1 gezeigten Regeln einfach sind, besteht ihr Nachteil darin, daß Sie sich hinsetzen und die Deklarationen klammern müssen. Statt die Zeit mit der Klammerung von »Unterdeklarationen« zu verschwenden, wäre es hilfreicher, die Theorie zu etwas Nützlicherem zu verallgemeinern. Die Regeln in Abbildung 4, Teile 1, 2, 3 und 5, erlauben Ihnen, Deklarationen im Fluge zu lesen.

Auf den ersten Blick scheint es, daß die Regeln komplexer sind, als die in Abbildung 1, aber es sind nur erweiterte Versionen dieser Regeln.

Nach der Bearbeitung einiger einfacher Deklarationen werden Sie sehen, wie natürlich diese Regeln sind. Sie sind nahezu identisch mit denen in Abbildung 1, da wir aber die Priorität der Operatoren kennen, ist es überflüssig, sie zu klammern. Wenn wir die Beispiele aus Abbildung 3 nehmen, kommen wir zu den Ergebnissen in *Abbildung 5.*

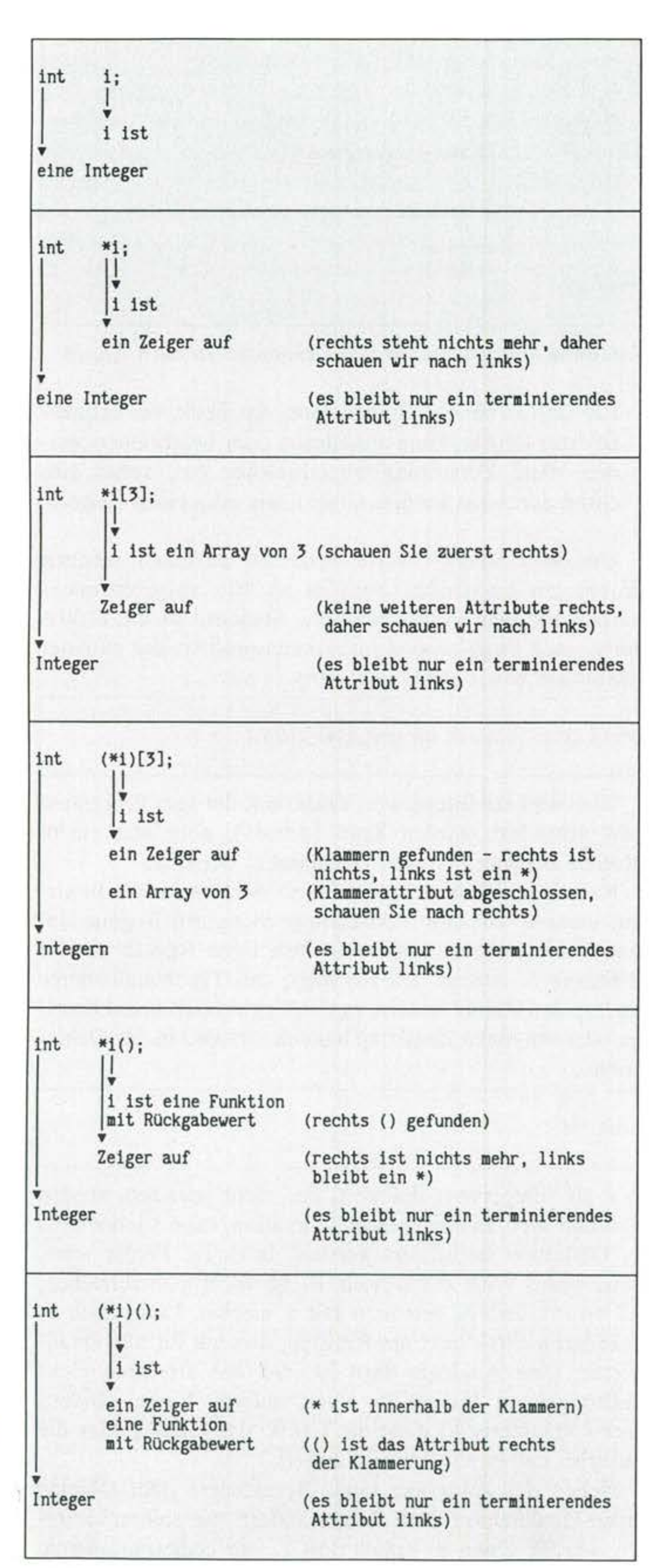


Abbildung 5: *Interpretation von Deklarationen nach Regeln*

```
int     **i;
        ||
        |i ist
        ein Zeiger auf        (rechts keine weiteren Attribute,
                               links ein *)
        einen Zeiger auf      (links noch ein *)
eine Integer

int     *(*i)();
         ||
         |i ist
         ein Zeiger auf       (rechts keine weiteren Attribute,
         eine Funktion                        links ein *)
         mit Rückgabewert     (Ende der Klammerung,
                               rechts ist ())
        Zeiger auf            (rechts keine weiteren Attribute
                               versuchen wir es links)
Integer

int     *(*i[])();
          ||
          i ist ein Array von (bleiben wir innerhalb der
                               Klammern, schauen wir nach
                               rechts)
         Zeigern auf           (jetzt links)
         Funktionen mit        (Ende der Klammerung
         Rückgabewert           jetzt Blick nach rechts)
        Zeiger auf
Integer
```

Abbildung 6: *Interpretation von Deklarationen nach Regeln*

Erzeugen wir einige Abkömmlinge der Deklarationen von K&R, erhalten wir die in *Abbildung 6* gezeigten. Benutzen wir Abkömmlinge der Beispiele der *Language Reference* des Microsoft C-Compilers Version 5.0, so erhalten wir die in *Abbildung 7* gezeigte. Wir wissen, daß diese Deklaration sich von der folgenden unterscheiden muß:

```
char *(*(abc()))[10]
```

Hier ist `abc` eine Funktion, die einen Zeiger auf einen Zeiger auf ein Array von 10 Zeigern auf `char` liefert, die auch so geschrieben werden kann:

```
char *(**abc())[10]
```

Zu guter Letzt zeigt *Abbildung 8* ein Beispiel einer `union`-Deklaration.

```
char    *(*(*i)())[10];

         i ist

         ein Zeiger auf eine Funktion mit Rückgabewert

         Zeiger auf ein Array mit 10

         Zeigern auf

char
```

Abbildung 7: *Interpretation von Deklarationen nach Regeln*

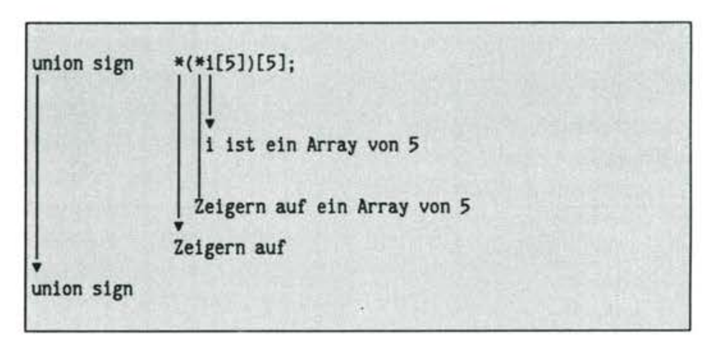


Abbildung 8: *Interpretation von Deklarationen nach Regeln*

Schreiben von Deklarationen

Die Regeln zum Schreiben von Deklarationen auf der Basis von deutschen Beschreibungen ist ebenso einfach, da es genauso wie das bisher Gesagte funktioniert, nur umgekehrt. Die Teile 1, 2, 4 und 5 der Abbildung 4 zeigen die zugehörigen Regeln. Die Beispiele in den Abbildungen 5, 6, 7 und 8 sind in umgekehrter Notation in *Abb. 9* dargestellt.

ANSI-Erweiterungen

Das X3J11-Komitee des ANSI-Instituts hat die K&R-Definition von C um Schlüsselworte und Nomenklaturen (wie Funktionsprototypen, die wir hier nicht besprechen) erweitert (*Abbildung 10*).

Die Schlüsselworte, die wir hier besprechen, sind `const` und `volatile`. Der Typenqualifizierer `const` gibt an, daß sich das mit dem Typ verbundene Objekt nicht verändert, das heißt, es wird ihm nichts zugewiesen und es wird nicht inkrementiert oder dekrementiert. Der Typenqualifizierer `volatile` gibt an, daß das Objekt entsprechend der C-Ausdrucksbewertung ausgewertet werden muß, was garantiert, daß C-Anweisungen und Objekte, die von diesen Anweisungen verwendet werden, einer vorgegebenen Ausführungsreihenfolge unterliegen. Da die Reihenfolge der Auswertung oftmals viele Optimierungen unmöglich macht, kann ein `volatile`-Objekt von einem anderen Programm als dem »besitzenden« verändert werden, ohne Furcht vor Inkonsistenz.

Einige Punkte zu diesen Schlüsselworten, die zu beachten sind:

- Einer `const`-Variablen kann kein Wert zugewiesen werden.
- Eine `const`-Variable, die nicht `volatile` ist, kann im ROM plaziert werden.
- Eine `volatile`-Variable kann durch Dinge wie DMA, asynchrone Prozesse oder gemeinsamen Speicher verändert werden.
- Eine `volatile`-Variable kann nicht wegoptimiert werden - das System muß der abstrakten C-Maschine folgen, soweit es diese Bezeichner betrifft.
- Ein Objekt ohne diese Attribute, das heißt, ein unqualifiziertes Objekt, kann ausgelesen oder beschrieben werden, ohne Zerstörung irgendwelcher Art, außer synchron durch ein anderes Objekt, wie etwa einen Zeiger.

Um diese Schlüsselworte besser zu verstehen, schauen wir uns ein praktisches Beispiel an. Ein ausgezeichnetes Beispiel aus dem vorgeschlagenen Standard ist die Deklaration eines Eingabeports im Arbeitsspeicher, der zu einer Echtzeituhr gehört. Dies sieht dann so aus:

```
extern const volatile int real_time_clock;
```

Hier wird ein Integerwert deklariert, der vom Programm nicht verändert werden kann (`const`), aber von einem externen Ereignis (`volatile`), nämlich der Uhr.

Nachdem wir wissen, was diese Schlüsselworte bewirken, müssen wir ihre Verwendung in unsere Regeln einbauen. Ein Anhang zu unseren bisherigen Regeln wird in *Abbildung 11* gezeigt. Daraus folgt, daß Typenqualifizierer den Typ des Objekt ändern, egal, ob es sich um einen Basistyp, oder um einen Zeigertyp handelt. So wird in der Deklaration

```
const int i;
```

`i` als Integerwert definiert, der nicht geändert werden kann und wird. In einer solchen Situation, kann `i` jedoch bei der Definition initialisiert werden, da es ein Fehler wäre, später einen Wert zuzuweisen. Es ist wichtig zu verstehen, daß `const` ändert, was man mit `i` machen kann, doch es verändert nichts direkt am Basistyp; dies gilt für alle Qualifizierer. Eine Analogie dazu ist, daß das Streichen eines großen grünen Hauses in einer anderen Farbe (ändern einer Characteristik) nicht die Größe des Hauses oder die Tatsache, daß es ein Haus ist, ändert.

Neben der Änderung eines Bezeichners oder Objekts könne Qualifizierer auch Zeiger ändern. Sie sollten verstehen, wie Sie einen so geänderten Zeiger codieren müssen, was wahrscheinlich der schwierigste Aspekt beim Lesen von Deklarationen ist.

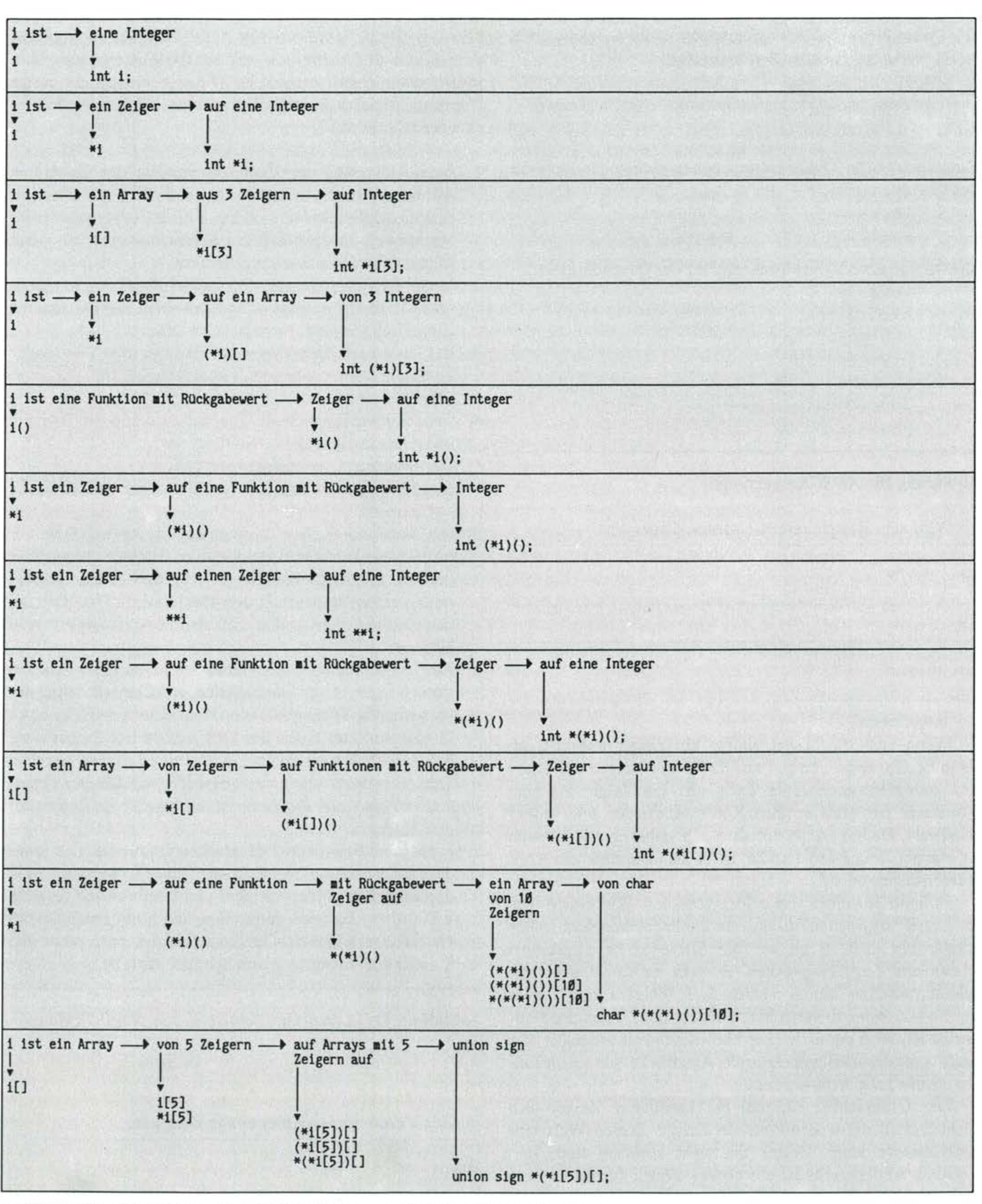


Abbildung 9: *Die Ableitung von Deklarationen aus deutschen Beschreibungen*

Die Syntax einer C-Deklaration wird von der angekündigten ANSI-Norm folgendermaßen festgelegt:

```
Speicherklasse Typ Qualifizierer Deklarator = Initialisierung;
```

Zusätzliche neue Qualifizierer können einer oder mehrere der folgenden sein:

```
const
volatile
```

Zusätzliche neue Typen können einer der folgenden sein:

```
void
signed char
unsigned char
unsigned int
unsigned long
long double
enum ...
```

Abbildung 10: *ANSI-Erweiterungen*

Wird zum Beispiel ein konstanter Zeiger als

```
int const *p
```

oder als

```
int *const p
```

geschrieben?

Wenn `int *p` besagt, daß `p` ein Zeiger auf `int` ist, dann besagt `int const *p`, daß `p` ein Zeiger auf einen konstanten Integerwert ist. Da `p` also ein Zeiger auf eine Konstante ist, muß es also Konstantenzeiger sein, oder? Vielleicht. Da bei `int * const p` `p` selbst eine Konstante ist, die ein Zeiger auf `int` ist, ist es also ebenfalls ein Konstantenzeiger, oder?

Es kann nur eines von beiden stimmen. Wir wollen dies im Detail besprechen, da uns ein ähnliches Problem später wieder begegnet, wenn wir die Microsoft-Erweiterungen (`near` und `far`) besprechen. Es geht darum, den Unterschied zwischen einem »konstanten Zeiger« und einem »`const`-Zeiger« zu verstehen. Obwohl der Unterschied nur gering ist, wird dadurch eine Mehrdeutigkeit aufgelöst, die auftritt, wenn verallgemeinernde Ausdrücke wie »Konstantenzeiger« gebraucht werden.

Der Unterschied ist, daß ein konstanter Zeiger sich nicht ändert; er ist konstant, ein Zeiger, dessen Wert sich nicht ändern kann. Zeiger die nicht konstant sind, sind variabel. Andererseits ist ein `const`-Zeiger (`const *`) ein Zeiger auf eine Konstante eines bestimmten Typs. Beachten Sie das Wort »konstant« im einen Fall und das Schlüsselwort `const` im anderen; sie sind nicht gleich.

Das Lesen und Schreiben von ANSI-Deklarationen gleicht dem Lesen und Schreiben von K&R-Deklarationen. Man muß jedoch einige zusätzliche Dinge beachten, da einige Typenqualifizierer hinzugefügt wurden. Dabei handelt es sich um folgendes:

1. Beim Vorschlag der Erweiterungen zu den Regeln in Abbildung 4 geht es uns darum, daß die ANSI-Erweiterungen Schlüsselwörter sind und nicht notwendigerweise Vorrangregeln gehorchen. Schlüsselwörter an sich folgen keinen konsistenten Mustern.
2. Unabhängig von anderen Typenspezifikationen hat jeder Qualifizierer (`const`, `volatile`) einen entsprechenden Zeigerttyp (`const *`, `volatile *`).
3. Bei einer fehlenden Typenspezifikation wird `int` angenommen, `const x` bedeutet zum Beispiel `int const x`. Hierbei handelt es sich jedoch um veralteten Stil.
4. Typenqualifizierer und Typenspezifikationen können ohne Rücksicht auf die Reihenfolge vermischt werden. Der jeweilige Ergebnistyp wird dadurch nicht verändert, `const int var` und `int const var` sind beispielsweise gleich.
5. Die Vermischung von Typenqualifizierern mit Deklaratoren (der Teil einer Deklaration, der den Bezeichner und die Attribute Funktion, Array und Zeiger angibt) *verändert* die Bedeutung der Deklaration. Die Bindung von Qualifizierern ändert sich also in Abhängigkeit vom Kontext.
6. Zur Verdeutlichung von 4. und 5. und um einen Fall wie `const int * p` klarzustellen, müssen wir folgendes vorschlagen: Typenqualifizierer verändern den Typ eines Deklarators. Im Fall einer Deklaration mit Zeigern wie bei *Typ * Typenqualifizierer Deklarator* (zum Beispiel `int * const var`) wird gesprochen *Deklarator Qualifizierer Zeiger auf Typ* (`var` ist ein konstanter Zeiger auf eine Integer).
7. Nicht qualifizierende Deklarationen (die mit der Standardqualifizierung) können so gelesen werden. Kein `const` bedeutet also variabel und kein `volatile` nicht `volatile`. Es wird empfohlen, die Standardqualifizierer zu einem Bezeichner hinzuzufügen, auch wenn die Standardqualifizierung kein Schlüsselwort ist.

Abbildung 11: *Lesen und Schreiben von ANSI-Deklarationen*

Zur Verdeutlichung hier einige Beispiele:

```
int i;
```

`i` ist eine variable Integer. Ihr können verschiedene Werte zugewiesen werden.

```
const int i;
```

i ist eine konstante Integer; ihr können nach der Deklaration keine Werte mehr zugewiesen werden.

```
int *p;
```

p ist ein variabler Zeiger auf eine variable Integer; sowohl p als auch der Integer, auf die der Zeiger zeigt, können Werte zugewiesen werden.

```
int * const p;
```

p ist ein konstanter Zeiger auf eine variable Integer; der Wert von p kann nicht geändert werden, die Integer, auf die er zeigt, kann verändert werden.

```
const int *p;
```

p ist ein variabler Zeiger auf eine konstante Integer. p kann verändert werden, der Integerwert, auf den der Zeiger zeigt, aber nicht.

```
int const *p;
```

p ist ein variabler `const`-Zeiger auf eine Integer. Da ein `const`-Zeiger ein Zeiger auf eine konstante Integer ist, ist p ein variabler Pointer auf eine konstante Integer, bedeutet also dasselbe wie `const int *p`.

```
const int * const p;
```

p ist ein konstanter Zeiger auf eine konstante Integer; weder der Zeiger p noch die Integer, auf die er zeigt, können geändert werden.

```
const *p;
```

p ist ein variabler Zeiger auf eine konstante Integer. Dies bestätigt `const int *p` und `int const *p`, denn es gibt keinen anderen Weg, es zu lesen. Es ist jedoch zu beachten, daß das Fehlen einer expliziten Typendeklaration nach der ANSI-Norm als schlechte Programmierweise angesehen wird, denn es führt zu Schludrigkeiten bei der Codierung von Deklarationen.

Gerüstet mit diesem Wissen, können Sie nun einige Varianten früherer Beispiele betrachten. Der Einfachheit halber werde ich in den folgenden Beispielen nur das Attribut `const` verwenden. Microsoft C erkennt das Schlüsselwort `volatile` nur syntaktisch, nicht semantisch. Wir werden auch das sogenannte Variabel-Attribut weglassen, da es nur dem Verständnis diente.

```
const int i;
```

i ist eine konstante Integer.

```
const int *i;
```

i ist ein Zeiger auf eine konstante Integer.

```
int * const i;
```

i ist ein konstanter Zeiger auf eine Integer.

```
const int * const i;
```

i ist ein konstanter Zeiger auf eine konstante Integer.

```
const int *i[3];
```

i ist ein Array von 3 Zeigern auf konstante Integer.

```
int * const i [3];
```

i ist ein Array von 3 konstanten Zeigern auf Integer.

```
const int * const i[3];
```

i ist ein Array von 3 konstanten Zeigern auf konstante Integer.

```
const int (*i)[3];
```

i ist ein Zeiger auf ein Array von 3 konstanten Integern.

```
int (* const i)[3];
```

i ist ein konstanter Zeiger auf ein Array von 3 Integern.

```
int (const * i)[3];
```

Dies ist ein Fehler, da das Schlüsselwort `const` nicht unmittelbar auf eine sich öffnende Klammer folgen kann.

```
const int (const * const i)[3];
```

Dies ist aus dem gleichen Grund ein Fehler.

```
const int (* const i)[3];
```

i ist ein konstanter Zeiger auf ein Array von 3 konstanten Integern.

```
const int *i();
```

i ist eine Funktion, die einen Zeiger auf eine konstante Integer zurückgibt.

```
int * const i();
```

i ist eine konstante Funktion, die einen Zeiger auf eine Integer zurückgibt. Viele Compiler nehmen sich die Freiheit, Qualifizierer bei Funktionen zu ignorieren. Zur Vereinfachung des Parsers erlaubt ANSI dies, ohne daß der Compiler ein Meldung ausgeben muß. Die ANSI-Norm definiert dies jedoch als »undefiniertes Verhalten«. Auf den ersten Blick scheint die Qualifizierung von Funktionstypen unsinnig zu sein, es kann jedoch dafür verwendet werden, eine Funktion im ROM zu plazieren.

```
const int (*i)();
```

i ist ein Zeiger auf eine Funktion, die eine konstante Integer zurückgibt.

```
int (* const i)();
```

i ist ein konstanter Zeiger auf eine Funktion, die eine Integer zurückgibt.

```
int (const * i)();
```

Dies ist ein Fehler, da das Schlüsselwort `const` nicht unmittelbar auf eine sich öffnende Klammer folgen kann.

```
const int **i;
```

i ist ein Zeiger auf einen Zeiger auf eine konstante Integer.

```
int ** const i;
```

i ist ein konstanter Zeiger auf einen Zeiger auf eine Integer.

```
int * const * i;
```

i ist ein `const`-Zeiger auf einen Zeiger auf eine Integer; i ist ein Zeiger auf einen konstanten Zeiger auf eine Integer.

```
int const * *i;
```

i ist ein Zeiger auf einen Zeiger auf eine konstante Integer.

```
int const * const * i;
```

i ist ein Zeiger auf einen konstanten Zeiger auf eine konstante Integer.

```
const int const * const * i;
```

Dies ist ein Syntaxfehler, da die ersten beiden `const`-Schlüsselworte zum Typ gehören und nicht zum ersten Stern.

```
const int * const * const * i;
```

i ist ein konstanter Zeiger auf einen konstanten Zeiger auf eine konstante Integer.

Sie werden in den obigen Beschreibungen, besonders bei `const int *i[3]`, `int * const i[3]` und `const int * const i [3]`, bemerkt haben, daß es hier nicht »konstantes Array«, sondern »Array von Konstanten« heißt. Das soll kurz erklärt werden.

Wenn beim Parsen einer Deklaration der letzte Typ ein Array ist und der aktuelle Typ ein Qualifizierer, so bezieht sich der Qualifizierer auf den Datentyp des Arrays, und nicht auf das Array selbst. Das macht Sinn, wenn Sie sich einige Ungereimtheiten von Arraytypen und Arrays vor Augen halten; besonders da ein Array nur eine Adresse repräsentiert und darüber hinaus nicht viele andere Attribute hat, da die Adresse einfach nur eine Speicherstelle ist. Tatsächlich wollen wir ja auch den Arrayelementen das Attribut `const` geben, da ihnen nichts zugewiesen werden soll. Zusätzlich sollten Sie bedenken, daß eine Arrayverwendung ebenso eine Konstante ist.

Microsoft-Erweiterungen

Ebenso wie ANSI hat es auch Microsoft für notwendig gehalten, die Sprache C um einige zusätzliche Schlüsselworte zu erweitern (*Abbildung 12*). Vier von ihnen sind Attribute von Funktionen.

Die ersten drei (`cdecl`, `pascal` und `fortran`) werden in Verbindung mit Funktionen in anderen Sprachen benötigt (BASIC, FORTRAN, MASM oder Pascal) oder ermöglichen den Aufruf von C-Funktionen aus diesen Sprachen. Diese speziellen Schlüsselworte sind eigentlich Compileranweisungen, die diesen veranlassen, speziellen Code zu erzeugen, so daß die übergebenen Parameter so behandelt werden, als ob sie in der Sprache geschrieben worden wären, die in der aufgerufenen Funktion verwendet wird.

Um eine C-Funktion zu deklarieren, die in Pascal geschrieben sein könnte, würden Sie folgendes verwenden:

```
extern int pascal func(long, int),
```

Microsoft hat die K&R- und ANSI-Definitionen um zwei zusätzliche Schlüsselwörter im Bereich der Qualifizierer erweitert:

```
far
near
huge
cdecl
pascal
fortran
interrupt
```

Abbildung 12: *Microsoft-Erweiterungen*

Ein Zeiger auf diese Funktion könnte so aussehen:

```
int (pascal *fp)(long, int);
```

Das Schlüsselwort `interrupt` ist auch eine Compileranweisung, die diesen veranlaßt, speziellen Code zu erzeugen. In diesem Fall wird Code erzeugt, um die Funktion mittels eines Interrupts aufrufen zu können.

Die drei anderen Schlüsselworte `near`, `far` und `huge` werden zur Adressierung eines Objekts auf Prozessoren mit segmentierter Architektur verwendet, wie die von Intel. Diese Schlüsselworte, und insbesondere `far`, werden oft bei der Programmierung in einem für diese Architektur typischen gemischten Speichermodell benötigt.

Microsoft behandelt die speziellen Schlüsselworte wie Deklaratorqualifizierer (*nicht* Deklarationsqualifizierer), das heißt, sie können nur syntaktische Einheiten auf der rechten Seite der Deklaration ändern, nicht aber den Basistyp, wie zum Beispiel `int`, direkt. Das Microsoft-C-Handbuch gibt an, daß diese Schlüsselworte nur den »Gegenstand« (item) unmittelbar rechts davon ändern, ähnlich wie das Schlüsselwort `const`. Die Beschreibung des Handbuchs ist in diesem Punkt aber unvollständig.

Wenn wir uns an die Diskussion der ANSI-Erweiterungen erinnern, bedeutet dies, daß die speziellen Schlüsselworte entweder einen Bezeichner oder einen Zeiger ändern; das heißt, sie ändern Objekte oder Zeiger auf Objekte. Es besteht jedoch ein Unterschied darin, wie diese behandelt werden, der nicht klar aus dem Handbuch hervorgeht. In Microsoft C ist die Syntax so, daß Zeiger von einem der speziellen Schlüsselworte geändert werden können. Daher ist eine Folge wie

```
int (far * p);
```

erlaubt, aber da ANSI eine solche Sytax verbietet, ist

```
int (const * p);
```

nicht erlaubt.

Daraus ergibt sich, daß Microsoft C eine Deklarationssyntax wie

```
MODIFIER * weitere_deklarations_informationen
```

akzeptiert, wogegen ANSI nur

```
* QUALIFIER weitere_deklarations_informationen
```

akzeptiert, wobei MODIFIER und QUALIFIER optionale Positionsparameter sind.

Bedenken Sie auch, daß `far` und `near` lange vor den ANSI-Erweiterungen `const` und `volatile` von Microsoft in die Sprache aufgenommen wurden. Der Grund für diesen unglücklichen Unterschied ist, daß ANSI die Syntax von Typenqualifizierern aus C++ (einer Obermenge von C, die objektorientierte Programmierung erlaubt) übernommen hat, die unabhängig zur selben Zeit entwickelt wurde, wie Microsoft seine Modifizierer eingeführt hat.

Das Gesagte über den Unterschied zwischen einem `const`-Zeiger und einen konstanten Zeiger gilt also auch hier. Bei diesen Attributen, zum Beispiel `far`, ist die Situation sogar noch mehrdeutiger, da far bereits ein englisches Wort ist. Ist `int *far p` ein Far-Zeiger (weil p als Zeiger im `far`-Speicher angelegt wird), oder ist `int far *p` ein `far`-Zeiger (da er auf eine `far`-Adresse zeigt)?

Wir klären dies in genau derselben Weise, wie wir es mit `const` erledigt haben. Wir bleiben einheitlich und erklären, daß `far *` (ein Schlüsselwort ändert einen Zeiger) `far`-Zeiger bedeutet. Im anderen Fall sagen wir, daß der Zeiger, der im `far`-Speicher liegt, ein entfernter Zeiger ist.

Ein letzter Punkt, der berücksichtigt werden muß, ist, daß die Speichermodell-Schlüsselworte von Microsoft C nicht mit automatischen Variablen verwendet werden können. Dies macht Sinn, da diese Variablen auf dem Stack geführt werden, und Sie keinen Einfluß darauf haben, ob der Stack `near` oder `far` ist. Dies bedeutet nicht, daß Sie keine lokalen statischen Variablen haben können, die `near` oder `far` sind; auch bedeutet es nicht, daß Sie keine automatischen Variablen verwenden können, die auf Daten in anderen Segmenten zeigen. Es bedeutet nur, daß eine automatische Variable selbst das Attribut nicht haben kann.

Hier einige weitere Beispiele, die diese Punkte unterstreichen:

```
int * far p;
```

p ist ein entfernter Zeiger auf eine Integer.

```
int far * p;
```

p ist ein `far`-Zeiger auf einen Integerwert. Da ein `far`-Zeiger ein Zeiger auf `far` ist, ist p ein Zeiger auf eine `far`-Integer.

Mitteilungen . . . . Mitteilungen . . . . Mitteilungen

Die Entwicklung des Sprachenmarkts

Der Sprachenmarkt hat sich seit 1975, als Microsoft das erste BASIC für den MITS ALTAIR herstellte, einschneidend verändert. Im folgenden werden die Entwicklungsrichtungen der Microsoft-Sprachenfamilie erörtert. Leitsatz dieser Bemühungen ist es, den Anwendern Werkzeuge zur Verfügung zu stellen, die ihren Bedürfnissen gerecht werden.

Sprachen: Grundlage des Microsoft-Erfolgs

Sprachen sind die Grundlage der Erfolgsgeschichte von Microsoft. Sie waren - mit dem ersten BASIC-Interpreter für den ersten im Handel erhältlichen PC - die Voraussetzung für die Gründung der Firma. Sie sind heute die Werkzeuge zur Entwicklung der Software, die Microsoft zum führenden PC-Softwareentwickler der Welt machte.

Seit 1975, als Bill Gates den ersten BASIC-Interpreter für den MITS ALTAIR geschrieben hat, veränderte sich der Sprachenmarkt gewaltig. In den Anfängen des Microrechner-Zeitalters waren Programmierer eine auserwählte Minderheit, die die innere Logik des Geräts, das sie programmierte, verstand und sich bequemte, Codezeilen zu schreiben, die nur wenige verstehen konnten. Heutzutage gibt es immer noch professionelle Programmierer, die auf dem Niveau der Assemblersprache arbeiten, aber die große Mehrheit befaßt sich mit höheren Programmiersprachen wie BASIC, C oder COBOL. Die wirklich spannenden Entwicklungen des Sprachenmarkts laufen allerdings nicht in diesem Bereich ab. Interessant wird es dort, wo sich eine riesige Anzahl von Ärzten, Rechtsanwälten, Börsen- und Immobilienmaklern und anderen Freiberuflern zum ersten Mal in die Heerscharen der Programmierer eingliedert.

Warum sollten Leute ihre eigenen Anwendungen in einem Umfeld (IBM PC, PS/2 und kompatible Systeme) schreiben, das buchstäblich zehntausende von Anwendungen zur Verfügung stellt? Die Antwort liegt in den hervorragenden Werkzeugen, mit denen PC-Anwender heutzutage ihre EDV-Probleme lösen können. Im folgenden soll diese Entwicklung auf dem Sprachenmarkt und die Herausforderung, der sich Microsoft stellen muß, um die bestmöglichen Werkzeuge zur Deckung des Anwenderbedarfs zu schaffen, betrachtet werden.

Die Marktentwicklung

Seit den Anfängen der Microcomputerprogrammierung hat sich der Sprachenmarkt in zwei unterschiedliche Richtungen entwickelt - eine für professionelle Programmierer und die andere für Leute, die Software schreiben, um ihre Arbeit besser durchführen zu können oder um ihre Computerkenntnisse zu vertiefen.

Die »Schnellprogrammierer«

Die »Schnellprogrammierer« stellen das Marktsegment mit der höchsten Wachstumsrate und der rasantesten Entwicklung dar. Es handelt sich hierbei um nichtprofessionelle Programmierer, die eine Programmiersprache als Werkzeug benutzen, um ihre Arbeit besser zu erledigen oder ihr Computersystem besser zu verstehen. Es überrascht nicht weiter, daß ihr Anforderungsprofil völlig verschieden von demjenigen der professionellen Programmierer ist. So unterschiedlich ihre Bedürfnisse sind, so unterschiedlich sind die Werkzeuge, die sie dafür benötigen.

Diese Anwendergruppe ist schon wegen ihrer Größe interessant. Da sich BASIC bei einem der englischen Sprache artverwandten Satzbau leicht erlernen läßt, entscheidet sich ein großer Teil dieser Menschen für Microsoft QuickBASIC. Bei einer Analyse der Kundengruppe, die in letzter Zeit Microsoft QuickBASIC gekauft hat, wurden einige interessante Fakten ans Tageslicht gebracht.

Zunächst einmal ergab sich, daß diese Menschen ein weit gefächertes Spektrum von Berufen ausüben, das vom Unternehmensberater über den EDV-Verwalter und Arzt bis zum Rechtsanwalt reicht. Gemeinsam ist ihnen die seriöse berufsmäßige Verwendung des PC als Arbeitswerkzeug. Wie nachstehend ersichtlich ist, haben sie keine typischen Einsteigersysteme:

CPU:	54% haben Systeme auf 80286- bzw. 80386-Basis
RAM:	96% verfügen über 640 Kbyte oder mehr
Festplatte:	96% haben mindestens 10 Mbyte Festplattenspeicherplatz
Maus:	61% verwenden eine Maus

Ihre Motivation, das Programmieren zu lernen, läßt sich mit einfachen Worten beschreiben: Sie haben ein Problem und sie müssen es lösen. Für viele ist die erste Lösung der Kauf eines Anwendungspakets, zum Beispiel für Tabellenkalkulation oder für die Datenbankverwaltung. Viele von ihnen sind jedoch mit der vom Anwendungspaket gebotenen Lösung nicht zufrieden oder stellen fest, daß die Leistungsfähigkeit des Pakets zu beschränkt ist. Sie beschließen daher, ihre eigene Lösung zu entwickeln.

Die von diesen Anwendern zu lösenden Probleme sind so vielfältig wie die Anwendungsmöglichkeiten des PC selbst:

Kundenspezifische Listengenerierung	72%
Kundenspezifische Datenbankverarbeitung	63%
Grafik (außer Präsentationsgrafik)	53%
aus Kalkulationstabellen und Datenbanken	52%
Buchführungssysteme	45%
Software-Entwicklungswerkzeuge	40%
Kommunikation	39%
Echtzeit-Prozeßsteuerung	37%
Präsentationsgrafik	26%
Computerspiele/Unterhaltung	25%
Interrupt-Bearbeitung	20%

Mitteilungen .... Mitteilungen .... Mitteilungen

Durch diese Untersuchungsergebnisse bekam Microsoft eine Vorstellung von den Problemen, die diese Kundengruppe zu lösen versucht. Die nächste Frage, die gestellt werden mußte, war: »Wie entwickeln wir ein Produkt, das diesen Bedürfnissen gerecht wird?« Während des ganzen letzten Frühlings und Sommers '88 wurden themenspezifische Workshops veranstaltet und Durchführbarkeitsstudien erstellt, um die subjektiven Hindernisse besser zu verstehen, die die Interessenten davon abhielten, das Programmieren zu erlernen. Die bei weitem häufigste Antwort auf die Frage »Warum haben Sie bisher nicht versucht, selbst zu programmieren?« lautete, man habe die Lernkurve anhand der heutigen Werkzeuge als zu lang und zu steil empfunden. Mit anderen Worten: der Einsatz, den diese Anwender benötigten, um produktiv zu sein, war zu hoch.

Professionelle Programmierer

Der heutige Berufsprogrammierer arbeitet in einem hart umkämpften Konkurrenzumfeld. Sowohl für diejenigen, die als gewerbliche Software-Anbieter arbeiten als auch für diejenigen, die als Firmenangestellte tätig sind, erfordert die Erstellung der schnellsten, kleinsten und sinnvollsten Anwendung Werkzeuge, die eine komplette Entwicklungslösung bieten.

Die Anforderungen des berufsmäßigen Programmiersprachen-Anwenders unterscheiden sich erheblich von denen des nicht berufsmäßigen Anwenders. Etwa 75% der C 5.0-Kunden sind berufsmäßige Programmierer, deren Haupteinnahmequelle das Schreiben von Software mit Microsoft C 5.0 ist. Sie schreiben mit C 5.0 unter anderem folgendes:

Software-Entwicklungswerkzeuge	59%
Kommunikation	56%
Kundenspezifische Listengenerierung	55%
Kundenspezifische Datenbankbearbeitung	55%
Interrupt-Bearbeitung	54%
Geräte-Treiber	40%
Grafik (außer Präsentationsgrafik)	36%
Statistische Analyse von Daten aus Kalkulationstabellen und Datenbanken	32%
Integrierte Systeme (ROM-fähiger Code)	30%
Buchführungssysteme	24%
Echtzeit-Prozeßsteuerung	39%
Präsentationsgrafik	20%

Diese professionellen Programmierer setzen Eigenschaften wie Leistungsfähigkeit, Betriebssicherheit und Herstellerrenommée an die Spitze ihrer Liste von Kaufkriterien. Sie setzen auch auf die neueste Technologie: 41% von ihnen schreiben Anwendungen für Microsoft Windows und viele von ihnen haben mit der Entwicklung von Anwendungen für MS OS/2-Systeme begonnen bzw. planen diese.

Die Herausforderung

Ein Programmiersprachenanbieter muß sich der Herausforderung stellen, daß er das bestmögliche Werkzeug schaffen muß, um den Bedürfnissen dieser beiden Anwendergruppen gerecht zu werden. Microsoft ist der Meinung, daß es unmöglich ist, ein einziges Produkt herzustellen, das den Bedürfnissen sowohl der berufsmäßigen Anwender als auch der Einsteiger gerecht wird. Dazu sind ihre Bedürfnisse, ihre Erwartungen und vor allen Dingen die Art, wie sie die Produkte anwenden, zu unterschiedlich. Daher hat Microsoft zwei sehr verschiedene Produktfamilien entwickelt, um den Bedarf dieser zwei Segmente zu decken. In beiden Fällen geht es primär darum, dem Kunden das bestmögliche Werkzeug zu bieten.

So ist zum Beispiel Benutzerfreundlichkeit für beide Segmente wichtig, wobei eine einfache, feste EDV-Umgebung mit unkompliziertem Editor und Testsystem für Anfänger durchaus ausreichen kann, aber eine weitaus differenziertere und flexiblere Programmumgebung für Programmentwickler benötigt wird, die eine Vielzahl von Werkzeugen, hoch konfigurierten (individualisierten) Editoren und leistungsstarken Testsystemen verwenden, um riesige Anwendungen zu schreiben, die auf verschiedenen Ebenen laufen (MS-DOS, Microsoft Windows, MS OS/2). Diese Unterschiede erfordern einen differenzierten Aufbau der Werkzeuge, bei dem eine klare Gewichtung der Verhältnisse zwischen Merkmalen wie Kompiliergeschwindigkeit und Codeausführungsgeschwindigkeit oder verwendetem Speicherplatz und Funktionalität erfolgt.

Der Microsoft-Leitgedanke bei den Quick-Sprachen ist es, den Anwender so schnell wie möglich über die Lernkurve zu führen. Dieser Vorgang wird als »Beherrschungsfreundlichkeit« bezeichnet. Microsoft QuickBASIC 4.0 ist das gegenwärtige Vorzeigeprodukt für diese Strategie - Nachfolgeprodukte sind in der Entwicklung. Bei Microsoft QuickBASIC 4.0 verknüpften sich die Bedienungsfreundlichkeit und die Dialogfähigkeit eines Interpreters mit der modernsten, am weitesten strukturierten BASIC-Version aller Zeiten.

Im Gegensatz zu den Lernwerkzeugen für den Einstiegsprogrammierer, entwickeln sich die professionellen Werkzeuge von Microsoft dem neuesten Stand der Technik entsprechend weiter und bieten damit eine komplette Entwicklungslösung für den Programmierer, dessen Einkommen von den Werkzeugen, die er verwendet, abhängt. Diese Entwicklung wird sich schwerpunktmäßig auf drei Bereiche konzentrieren.

Die Entwicklung von Schlüsseltechnologien

Microsoft ist ein Unternehmen, das auf der Grundlage von Technologie aufgebaut wurde. Ob bei Betriebssystemen, Programmiersprachen oder Anwendungen, die grundlegende Produkttechnologie muß den neuesten Stand der Technik darstellen. Bei Programmiersprachen betrifft dies

Mitteilungen . . . . Mitteilungen . . . . Mitteilungen

Produkte wie Codegeneratoren, Testsysteme und sonstige Werkzeuge, EDV-Umgebungen, die die Entwicklung von Software durch Arbeitsgruppen unterstützen sowie objektorientiertes und visuelles Programmieren für grafische Anwenderinterfaces wie Microsoft Windows und den Presentation Manager.

Zunehmende Produktivität der Programmierer

Berufsmäßige Programmierer mußten sich bisher mit dezentralen Werkzeugen zufriedengeben, die wenig oder gar keine Integration zuließen. Eine wirklich fortschrittliche, integrierte Entwicklungsumgebung muß flexibel genug sein, um das breite Spektrum an Werkzeugen, das die Profis installieren möchten, zu handhaben, und leistungsstark genug sein, um die riesigen Anwendungen, die entwickelt werden, zu bearbeiten.

Rechtzeitige Unterstützung von neuen Betriebssystemen und neuer Hardware

Professionelle Programmentwickler verlangen Werkzeuge von einer Leistungsfähigkeit, die die Ausnutzung neuer Hardware und neuer System-Softwareebenen ermöglicht. Mit dieser Unterstützung kann der Profi rasch in den heiß umkämpften Software-Entwicklungsmarkt eindringen.

Unsere Antwort auf die Herausforderung

Microsoft entwirft jede Quick-Sprache und jede professionelle Sprache mit einem Ziel: Das beste Werkzeug für den jeweiligen Anwenderkunden zu entwickeln. Bei den professionellen Sprachen bedeutet dies die Erstellung einer kompletten Entwicklungslösung, das heißt von Werkzeugen, die die Leistungskraft bieten, welche der berufsmäßige Programmentwickler benötigt. Bei den Quick-Sprachen bedeutet es, neue Maßstäbe in der »Beherrschungsfreundlichkeit«, dem Abkürzen der Lernkurve und dem raschen Ermöglichen von Anwenderproduktivität zu setzen.

Entwickler des VMS-Betriebssystems kommt zur Microsoft-»Operating Systems Group«

Der Entwickler einiger der erfolgreichsten Betriebssysteme der Digital Equipment Corporation, David Cutler, ist zur Microsoft Corporation gewechselt, um dort ein Entwicklungs-Team für fortschrittliche Betriebssystem-Software zu leiten.

Cutler, der 17 Jahre lang bei Digital Equipment tätig war, hat an der Entwicklung und Implementierung des RSX-11M-Betriebssystems für die PDP-11-Computer - das Digital Equipment in den 70er Jahren zum führenden Unternehmen bei Echtzeit-Systemen gemacht hat - und an der Entwicklung des VMS-Betriebssystems für die VAX-Computer - mit dem Digital in den 80er Jahren zum Marktführer bei Minicomputern wurde - wesentlich mitgewirkt.

Zusätzlich zu seiner Tätigkeit bei der Entwicklung von Betriebssystemen war Cutler auch an der Entwicklung von Computer-Architekturen und Hardware beteiligt. Sein Team erstellte bei DECwest in Seattle den MicroVAX-Computer, Digitals erste Implementierung der VAX-Architektur auf einem Mikrocomputer. Cutler hat außerdem umfangreiche Erfahrungen in der Compiler-Entwicklung durch die Implementierung von PL/1-, C- und anderen Sprachen-Compilern.

David Cutler war einer von drei »Senior Corporate Consultant« in der gesamten Digital Equipment Corporation. Bei Microsoft leitet er ein Team, das sich auf die Entwicklung zukünftiger Versionen von Microsoft-OS/2 und anderer fortschrittlicher Betriebssystem-Software konzentriert.

Steve Ballmer, Vice President of Systems Software bei Microsoft, sagte anläßlich des Wechsels von David Cutler, daß Microsoft außerordentlich erfreut darüber ist, einen Mann mit den Fähigkeiten und Erfahrungen David Cutlers nun in den eigenen Reihen zu haben. Die Einstellung David Cutlers durch Microsoft zeige außerdem deutlich das Engagement für MS OS/2 als Software-Plattform der Zukunft. »David ist als jemand anerkannt, der große Projekte leiten und sie pünktlich und in höchster Qualität ausführen kann«, fügte Ballmer hinzu.

David Cutler sagte zu seinem Start bei Microsoft, daß er darin eine einzigartige Möglichkeit sieht, seine berufliche Karriere in einen Bereich der Computertechnik auszudehnen, in dem er bisher noch nicht gearbeitet hat. Er sieht sich selbst mehr als Leiter eines Teams denn als Manager und ist der Meinung, daß eine gute Leitung ausschlaggebend für erfolgreiche Entwicklungsteams ist.

MS OS/2 jetzt mit Presentation Manager

Microsoft und IBM kündigten vor kurzem, wie versprochen, die Auslieferung der gemeinsam entwickelten Version 1.1 des Betriebssystems OS/2 mit dem Presentation Manager an. Der Presentation Manager, die grafische Bedienungsoberfläche für das OS/2-Betriebssystem, wurde mit der OS/2-Version 1.0 im April 1987 angekündigt. IBM begann mit der Auslieferung der Version 1.0 von OS/2 im Dezember 1987. Die jetzige Vorstellung ist das zweite große Release der Standardversion von OS/2. IBM kündigte die sofortige Auslieferung der OS/2-Version 1.1 an. Microsoft wird mit seinen anderen OEM-Kunden daran arbeiten, MS OS/2 in der Version 1.1 für die Computer-Hardware dieser Hersteller auszulegen. Es wird erwartet, daß die anderen Computer-Hersteller mit der Auslieferung von OS/2, Version 1.1, Anfang 1989 beginnen.

Im Rahmen einer Presseveranstaltung in New York zeigten IBM, Microsoft und eine Reihe anderer Software-Hersteller neuentwickelte Produkte für den Presentation Manager. Folgende Unternehmen demonstrierten neue

Mitteilungen .... Mitteilungen .... Mitteilungen

Produkte oder kündigten die Unterstützung von OS/2, Version 1.1, an: Aldus, Ashton Tate, Borland, Computer Associates, Digital Research, IBM, Lotus, mdbs, Micrographx, Microrim, Software Publishing, 3Com, Versacad, WordPerfect und Xcellenct. Microsoft selbst zeigte in New York MS-Excel für den Presentation Manager.

OS/2 ist nicht nur die Betriebssystem-Plattform für eine ganz neue Generation von Anwendungs-Software, es ist in gleichem Maße die Basis für wichtige Neuentwicklungen bei lokalen Netzwerken, Kommunikations- und Datenbank-Produkten, wie zum Beispiel IBM-Extended Edition, Microsoft-LAN-Manager, Ashton-Tate/Microsoft-SQL-Server sowie weiterer ähnlicher Produkte.

MS OS/2 wird nach Meinung von Christian Wedell, Geschäftsführer der deutschen Microsoft GmbH, einen außerordentlich starken Einfluß auf den PC-Markt haben, weil in einem vergleichbaren Zeitraum sehr viel mehr Computer-Hersteller OS/2 anbieten und viel mehr Anwendungen verfügbar sein werden, als für MS-DOS. Ein großer Anteil dieser Aktivitäten richtet sich auf die Entwicklung von Anwendungs-Software für den MS OS/2-Presentation Manager. Microsoft sieht deshalb die planmäßige Auslieferung der OS/2-Version 1.1 als einen wichtigen Meilenstein auf dem Wege, neue Anwendungs-Software auf den Endanwender-Markt zu bringen.

OS/2 mit Presentation Manager bietet eine volle Unterstützung für den Protected-Mode-Betrieb von PCs auf 80286- und 80386-Prozessor-Basis. Den Anwendern stehen nun 16 Mbyte physischer Hauptspeicher und 1 Gbyte virtueller Speicher über ein Standardgrafik-Bedienungsinterface zur Verfügung, das die OS/2-Schlüsselfunktionen, wie zum Beispiel Multitasking, einfach bedienbar machen.

Ein wichtiges Merkmal für Software-Entwickler ist das hochentwickelte, prioritätsgesteuerte, wahlfreie Steuerprogramm (Preemptive Scheduler), das durch einen umfangreichen Satz von Interprozeß-Kommunikationsmöglichkeiten unterstützt wird. Anwendungen lassen sich als eng zusammenarbeitende Gruppen von Tasks entwickeln, woraus sich eine verbesserte Effizienz und die Basis für echte Netzwerk-Anwendungen ergibt.

MS-Excel erstes Anwendungsprogramm für den Presentation Manager

Microsoft zeigte in diesen Tagen mit Excel für den Presentation Manager die erste Anwendungs-Software, die die Version 1.1 des Betriebssystems MS OS/2 unterstützt. Microsoft Excel ist das erste Tabellenmanagement-Programm, bei dem die Vorteile der grafischen Bedienungsumgebung genutzt werden, um hohe Funktionalität, zahlreiche Darstellungsmöglichkeiten und gute individuelle Anpaßbarkeit zu bieten. Microsoft Excel ist nicht nur das führende Tabellenkalkulations-Programm auf dem Apple-Macintosh, es gewinnt auch in der Windows-Version mit mehr als 20.000 im Monat weltweit verkauften Paketen einen schnell wachsenden Anteil im MS-DOS-Markt.

Obwohl Microsoft noch keinen genauen Termin und Preis für die MS OS/2-Version von Microsoft Excel genannt hat, wurde bestätigt, daß bisherige Besitzer von Microsoft Excel für Windows zu einem günstigen Upgrade-Preis auf Microsoft Excel für den Presentation Manager umsteigen können.

Die Entwicklung der MS OS/2-Version von Microsoft Excel ist derzeit schon weiter gediehen als geplant war. Microsoft entwickelt schon seit mehreren Jahren Anwendungsprogramme für Microsoft Windows und für den Macintosh und hat mit grafischen Bedienungsumgebungen mehr Erfahrung als andere Hersteller von Anwendungs-Software. Die Entwicklung von Microsoft Excel für MS OS/2 geht reibungslos voran, so daß zu erwarten ist, daß Microsoft Excel das erste Tabellenkalkulations-Programm sein wird, das für den Presentation Manager verfügbar ist.

Zusätzlich zur Familie der Microsoft Windows-Produkte plant Microsoft eine komplette Linie von Anwendungsprogrammen für den Presentation Manager. Da bei den Anwendungsprogrammen mit einer einheitlichen »Core Engine of Code« gearbeitet wird, müssen Microsoft Windows-Anwendungen nicht vollständig für MS OS/2-1.1 umgeschrieben werden. Etwa 80 Prozent des Programmcodes lassen sich auch unter MS OS/2 einsetzen. Alle geplanten Anwendungsprogramme von Microsoft werden mit einem gemeinsamen Bedienungs-Interface ausgestattet sein, so daß die Trainings- und Supportkosten reduziert werden und die Anwender auf einfachere Weise mit mehreren Anwendungen arbeiten können.

Excel für den Presentation Manager zeigt die Vorzüge von MS OS/2

Microsoft Excel für den Presentation Manager wird mit Microsoft Excel 2.1, der derzeitigen Windows-Version, vieles gemein haben. Beide Versionen entsprechen dem IBM-CUA-Standard (Common User Access) und haben das gleiche Interface. Anwender, die auf die neue Version umsteigen wollen, müssen deshalb kein neues Programm lernen. Beide Versionen haben darüber hinaus auch identische Dateiformate, so daß keine Daten geändert werden müssen. Neben der Verringerung der Trainingskosten wird Microsoft Excel für den Presentation Manager auch die anderen Nutzeffekte bieten, die mit MS OS/2 verbunden sind.

MS OS/2 ermöglicht es, daß Microsoft Excel vollständig in den Arbeitsspeicher geladen werden kann. Ein Hin- und Herladen zwischen Arbeitsspeicher und Massenspeicher ist deshalb nicht nötig. Zahlreiche Tasks werden sehr viel schneller laufen, zum Beispiel das Drucken, die grafische Darstellung, Makros und der Bildschirmaufbau. Die Anwender werden außerdem in der Lage sein, sehr viel größere Tabellen zu erstellen, ohne Speichererweiterungs-Platinen oder Treiber zu benötigen. Der Presentation

Mitteilungen .... Mitteilungen .... Mitteilungen

Manager macht es möglich, daß zwischen Microsoft Excel und anderen MS OS/2-Anwendungen einfach hin- und hergeschaltet werden kann. MS OS/2 erlaubt echten Multitasking-Betrieb, so daß Microsoft Excel zum Beispiel im Hintergrund arbeiten kann, während der Anwender auf dem Bildschirm eine andere Anwendung laufen läßt. Die Presentation Manager-Version von Microsoft Excel wird nicht zuletzt auch bei Netzwerk-Betrieb eine wesentlich höhere Leistung bieten, weil sowohl der Netzwerk-Treiber als auch Microsoft Excel komplett in den Hauptspeicher ladbar sind, so daß beim Betrieb nicht immer wieder auf die Festplatte zugegriffen werden muß. Die Anwender von Microsoft Excel für den Presentation Manager werden außerdem feststellen, daß die Erstinstallation einfacher ist, weil Speichertreiber und Festplatten-Zugriffs-Utility nun nicht mehr notwendig sind.

Microsofts Pläne zur Drucker-Unterstützung für den Presentation Manager

Unlängst gab Microsoft bekannt, daß eine umfassende Unterstützung für eine breite Palette von Druckertreibern unter MS OS/2-Presentation Manager geplant ist. Die Treiber-Software wird zusammen mit der Firma Bauer Enterprises, San Jose, Kalifornien, entwickelt und soll Bestandteil der Standardversion MS OS/2-1.1 sein. Sie wird darüber hinaus allen MS OS/2-OEM-Lizenznehmern zur Verfügung stehen.

Die Druckertreiber-Software mit der Bezeichnung »Generic Printer Driver« unterstützt eine breite Palette von Geräten, wie Matrix-, Typenrad- und Laserdrucker. Die Unterstützung für Laserdrucker schließt auch Drucker mit HP-PCL-Emulation ein. Die »Generic Printer Driver«-Software vereinfacht insbesondere den Aufwand, den die Druckerhersteller üblicherweise treiben müssen, um eine entsprechende Unterstützung ihrer Produkte durch das Betriebssystem zu gewährleisten. Im wesentlichen entwickelt der Druckerhersteller nun nur noch eine einzige Tabelle, als Drucker-Beschreibungstabelle bezeichnet, die die Merkmale und Funktionen des Geräts spezifiziert. Die Informationen dieser Tabelle werden dann durch den »Generic Driver« geladen und zur Ansteuerung des jeweiligen Druckers benutzt. Dieses Verfahren entbindet den Druckerhersteller von der erheblich komplexeren Aufgabe, einen speziellen MS OS/2-Treiber für einen bestimmten Drucker zu entwickeln.

Die Firma Bauer Enterprises wird den Druckerherstellern ein Entwicklungs-Kit zur Verfügung stellen, das die Entwicklung und den Test von Drucker-Beschreibungstabellen ermöglicht. Darüber hinaus bietet Bauer Enterprises einen Prüf- und Wartungsservice an, der die Fehlerfreiheit der Drucker-Beschreibungstabellen und eine rechtzeitige Erweiterung für neue Druckermodelle sicherstellen soll. Die Generic-Druckertreiber-Software wird für Microsoft-OS/2-OEM-Kunden im ersten Quartal 1989 verfügbar sein.

»Microsoft entschied sich auf diesem diffizilen Feld für eine Zusammenarbeit mit Bauer Enterprises, weil dieses Unternehmen eine reiche Erfahrung in der Entwicklung von Drucker-Software hat«, so Christian Wedell, Geschäftsführer der deutschen Microsoft GmbH, zu der Kooperation mit Bauer Enterprises.

Bei Bauer Enterprises geht Cal Bauer, Gründer und Präsident des Unternehmens, davon aus, daß die Zusammenarbeit mit Microsoft eine hervorragende Nutzung des Know-hows ermöglicht, das sein Unternehmen in puncto Druckertreiber-Software besitzt. Bauer ist außerdem der Meinung, daß MS OS/2 ein führendes Betriebssystem im Bürocomputer-Bereich wird und daß die Generic-Druckertreiber-Software eine ausgezeichnete Lösung für eine große Zahl von Druckerherstellern ist.

Zahlreiche führende Druckerhersteller planen die Unterstützung ihrer Drucker in Bezug auf den MS OS/2-Presentation Manager und stehen zur Zeit in der Diskussion über die Generic-Druckertreiber-Software mit Microsoft und Bauer. Diese Druckerhersteller repräsentieren mehr als 80 Prozent des weltweiten PC-Druckermarkts und schließen Firmen wie Alps America, Brother International, Citizen, Dataproducts Corporation, Fujitsu, Epson, NEC Information Systems, Office Automation Systems Inc., Okidata, Olivetti, Ricoh, TEC und Toshiba ein.

Steve Lapinski, Vize-Präsident für Marketing bei Epson, äußerte, daß der MS OS/2-Presentation Manager eine wirklich neue Betriebssystem-Umgebung für den heutigen Anwender darstellt und daß Epson plant, den Presentation Manager für seine umfassende Druckerpalette voll zu unterstützen.

Antonio Maccari, Manager für Forschung und Entwicklung bei Olivetti, sagte, daß Olivetti kürzlich die Bauer-Technologie für seine PostScript-kompatiblen Drucker übernommen hat und man im Hause Olivetti der Meinung ist, daß die Kooperation zwischen Microsoft und Bauer Enterprises ein wichtiger Faktor für die Etablierung eines Standard-Druckerinterfaces innerhalb von MS OS/2-Presentation Manager ist. Olivetti lege sich auf den neuen Standard fest, fügte Maccari hinzu.

Bruce Friesen, Direktor für das Drucker-Marketing der Information Systems Division von Toshiba Amerika, führte aus, daß Toshiba Amerika die MS OS/2-Presentation Manager-Druckersupport-Strategie voll unterschreibt. Toshiba Amerika erwarte von der Generic-Druckertreiber-Software erhebliche Einsparungen an Zeit und Ressourcen. »Es ist gut zu wissen, daß die Toshiba-Drucker alle Anwendungen komplett unterstützen werden, wenn der MS OS/2-Presentation Manager auf den Markt kommt«, setzte Friesen hinzu.

Alex Schibanoff, Marketing Direktor der Brother International, vertrat die Ansicht, daß die Entwicklung eines Industrie-Standard-Druckertreibers für den MS OS/2-Pre-

Mitteilungen .... Mitteilungen .... Mitteilungen

sentation Manager ein Beispiel dafür ist, daß Microsoft die führende Position im Bereich der PC-Software erreicht hat. Schibanoff ist außerdem der Meinung, daß sowohl PC-Anwender, Software-Entwickler und Druckerhersteller einen Nutzen durch die neue Treiber-Software haben.

Die Bauer Enterprises wird neben dem Generic-Treiber für den MS OS/2-Presentation Manager einen äquivalenten Treiber für Microsoft-Windows entwickeln. Das Format der Drucker-Beschreibungstabellen für MS-Windows wird identisch zu dem für den MS OS/2-Presentation Manager sein. Dies bedeutet gleichzeitig eine weitere Reduzierung des Aufwands, den die Druckerhersteller bei der Entwicklung von Treiber-Software für ihre Produkte treiben müssen. Microsoft hat die Absicht, den Generic-Treiber in zukünftige Versionen von MS-Windows zu integrieren.

Für die Hewlett-Packard Company, die Hersteller einer Reihe weitverbreiteter PC-Drucker einschließlich des HP-LaserJets-II ist, kommentierte William P. McGlynn, Marketing Manager der HP Boise Printer Division, die Entwicklung der Generic-Treiber-Software mit den Worten: »Hewlett-Packard sieht den MS OS/2-Presentation Manager als den kommenden Standard! Wir arbeiten eng mit Microsoft zusammen, um den Anwendern dieser Betriebssystem-Umgebung hochqualitative Druckerlösungen von Hewlett-Packard bieten zu können.«

Hintergrund-Informationen zu Geräte-Treibern

Geräte-Treiber sind Software-Komponenten, die der Kommunikation zwischen dem Betriebssystem und Hardware-Einheiten, wie Disketten-/Festplatten-Laufwerken, Bildschirmen und Druckern dienen. Sie bilden eine Trennstelle zwischen dem Betriebssystem sowie der Anwendungs-Software auf der einen Seite und den Hardware-Einheiten auf der anderen Seite. Unter dem MS OS/2-Presentation Manager arbeitet die Anwendungs-Software in einer geräteunabhängigen Weise, die es möglich macht, daß zum Beispiel ein Dokument ohne weiteres auf verschiedenen Druckern ausgegeben werden kann. Die Anwendungs-Software benötigt dafür keine Informationen über die physischen Merkmale der jeweiligen Drucker.

Bauer Enterprises hat seit seiner Gründung im Jahre 1985 Treiber-Software für Microsoft entwickelt und getestet. Das Unternehmen ist führend in der Drucker-Software-Technik und bietet Druckerherstellern eine komplette Palette von Software-Werkzeugen und Dienstleistungen an. Bauer beschäftigt sich mit Anwendungs-Software, Kompatibilitätstests, Anwendungstreiber-Entwicklung und der Entwicklung von Druckersprachen. Die Firma hat Betriebsstätten in San Jose, Kalifornien, sowie in Taipeh, Taiwan, und beschäftigt 40 Mitarbeiter, von denen 27 Software-Ingenieure sind.

Microsoft und Hewlett-Packard portieren den MS OS/2-Presentation Manager auf Unix

Microsoft, Santa Cruz Operation (SCO) und Hewlett-Packard haben kürzlich eine Vereinbarung zur gemeinsamen Entwicklung einer Unix-Version des MS OS/2-Presentation Managers bekanntgegeben. Doug Michels, Mitbegründer und Vize-Präsident von SCO, erklärte anläßlich dieser Ankündigung vor der Presse: »Der *Presentation Manager/X* ist kompatibel mit *X Window*-Anwendungen und erlaubt es Anwendern, sowohl X- als auch Presentation Manager-Anwendungen mit dem selben Interface zu nutzen. Damit kommen sich MS-DOS bzw. MS OS/2 und SCO-Unix aus Anwendersicht ein gutes Stück näher - eine Entwicklung, die Microsoft und Santa Cruz Operation seit Jahren betreiben.«

Der Presentation Manager/X stellt eine Erweiterung des *Common X Interface* (CXI) dar, das ebenfalls in diesen Tagen angekündigt wurde, und ist der nächste Schritt in Richtung Anwendungsportabilität bei verschiedenen Betriebssystemen. Das grafische Bedienungsinterface CXI verleiht der Unix-Bedienungsoberfläche dasselbe Aussehen und die Funktionen, wie sie von PCs bekannt sind, die unter MS-DOS mit Microsoft Windows oder MS OS/2 mit dem Presentation Manager laufen. Außerdem hat der Presentation Manager/X Anwendungsprogramm-Interfaces (API), die denjenigen des MS OS/2-Presentation Manager entsprechen. Der Presentation Manager/X bietet zwei wesentliche Vorzüge:

1. In Verbindung mit CXI ermöglicht er es Anwendern, die mit Microsoft Windows oder mit MS OS/2-Presentation Manager vertraut sind, auch auf entsprechenden Unix-Computern ohne großen Lernaufwand zu arbeiten.
2. Der Anwender kann auf einfache Weise Anwendungen von MS OS/2 1.1 auf Unix und umgekehrt transportieren.

»Unix-Software-Entwickler wollen eine stabile grafische Betriebssystem-Umgebung haben, die drei Merkmale aufweist: eine gute Bedienungs-Interface-Technik, die Unterstützung durch einen führenden Hersteller sowie eine reibungslose Portiermöglichkeit zwischen OS/2 und Unix. Der Presentation Manager/X hat alle drei Merkmale«, so Christian Wedell, Geschäftsführer der Microsoft GmbH.

Microsoft, Santa Cruz Operation und Hewlett-Packard entwickeln den Presentation Manager/X als Reaktion auf Marktanforderungen. Da der Presentation Manager/X sowohl auf Intel- als auch auf Nicht-Intel-Hardware eingesetzt werden kann, ergibt sich für die Entwickler von Anwendungs-Software ein großer Zielmarkt.

Hewlett-Packard ist der erste führende Computer-Hersteller, der bekanntgab, daß er den Presentation Manager/X einsetzen wird. Er wurde außerdem der Open-Software-Foundation (OSF) als Reaktion auf die kürzliche Ausschreibung der OSF für zukünftige Erweiterungen des Common X Interfaces unterbreitet.

Mitteilungen . . . . Mitteilungen . . . . Mitteilungen

Der Presentation Manager/X ist so ausgelegt, daß er mit dem Industrie-Standard X-Window-System koexistieren kann. Da das X-Window-System die Basis für CXI ist, werden CXI- und Presentation Manager/X-Anwendungen gleichzeitig auf dem Bildschirm laufen können.

Die Verfügbarkeit und der Preis für den Presentation Manager/X-Toolkit werden in der ersten Hälfte dieses Jahres bekanntgegeben.

Microsoft liefert MS-DOS 4.01 OEM-Paket aus

Microsoft gab kurz vor Jahreswechsel bekannt, daß die Auslieferung des Software-Pakets MS-DOS 4.01 zusammen mit der MS-DOS-Shell an die Hersteller von Personalcomputern begonnen hat. Diese Version, mit dem Namen *MS-DOS Betriebssystem Version 4.01*, ist vollständig kompatibel mit PC-DOS 4.01 und beinhaltet eine verbesserte Unterstützung der »Expanded Memory Specification« (EMS). PC-DOS 4.01 ist die aktuelle DOS-Version, welche die IBM derzeit zusammen mit ihren PCs ausliefert.

Das MS-DOS 4.01 System bietet eine DOS-Bedienungsoberfläche, die es als grafisches Datei-Management-System dem Anwender erlaubt, über die grafische Eingabe Festplatten-Dateien zu handhaben, Anwendungen zu starten und die Basis-DOS-Funktionen per Pull-Down-Menü und Dialog-Fenster auszuwählen. Bisherige MS-DOS Versionen hatten ein zeichen-/zeilenorientiertes Interface (Prompt Command Line), bei dem der Anwender die Befehle als definierte Zeichenketten eingeben mußte.

»Die DOS-Shell bietet MS-DOS Anwendern die Vorteile einer grafischen Bedienungsoberfläche, wobei der Anwender nun mit dem MS-DOS-4.01-Interface auf vergleichbare Weise arbeiten kann wie auf höherer Ebene mit dem MS OS/2 Presentation Manager, so Christian Wedell, Geschäftsführer der deutschen Microsoft GmbH, zur Auslieferung von MS-DOS 4.01.

Die Reaktion der Anwender auf die neue grafische DOS-Oberfläche ist nach Meinung von Jerry Schneider, ehemaliger Präsident und Gründungsmitglied der Capital PC User Group Inc., positiv, weil die DOS-Shell die Leistung und Vielseitigkeit des Betriebssystems für den Anwender erschließt, ohne daß er ein umfangreiches Wissen über Befehle und deren Syntax haben muß. Auch Anfängern ermöglicht die neue DOS-Shell durch die einfache Bedienbarkeit schnell die ganze Leistungsfähigkeit des Betriebssystems zu nutzen.

Ein weiteres Merkmal der MS-DOS Version 4.01 ist der LIMulator, ein Treiber, der in PCs auf 80386-Basis die Nutzung eines erweiterten Hauptspeichers ermöglicht. Darüber hinaus bietet die MEM-Utility in MS-DOS 4.01 dem Anwender die konventionelle Adressierung des Hauptspeichers. Die neue Betriebssystemversion unterstützt Festplattenpartitionen mit einer Größe von mehr als 32 Mbyte.

Das mit Microsoft-Label versehene Software-Paket aus MS-DOS 4.01 und MS-DOS-Shell wird direkt an die Hardware-Kunden geliefert. Wie auch schon bei früheren Versionen, handelt es sich bei MS-DOS 4.01 um ein reines OEM-Lizenzprodukt, das von Microsoft nicht direkt an Endkunden bzw. Händler ausgeliefert wird. Microsoft geht davon aus, daß OEM-Lizenznehmer im ersten Quartal 1989 damit beginnen, modifizierte MS-DOS 4.01 Versionen auszuliefern. Der Vertrieb der MS-DOS 4.01 US-Version erfolgt ausschließlich über Hardware-Hersteller.

Textverarbeitung auf dem Macintosh mit Word 4.0 optimiert

Microsoft stellte in diesen Tagen die neue Version des Textverarbeitungs-Programms Microsoft Word 4.0 für den Macintosh vor, das den Anwendern große Leistungsfähigkeit und Flexibilität im gesamten Bereich der professionellen Textverarbeitung ermöglicht. Das Programm bietet eine Vielzahl individueller Spezifikationen, einfache Handhabung und Kompatibilität zu anderen Software-Produkten.

»Die umfangreiche Erweiterung der neuen Version macht deutlich, daß Microsoft auf die Wünsche der Macintosh-Anwender eingegangen ist und die Gelegenheit wahrgenommen hat, zahlreiche Funktionen in das Programm aufzunehmen, die von den Anwendern gewünscht wurden«, erläutert Microsoft-Geschäftsführer Christian Wedell.

Microsoft Word 4.0 ist mit speziellen Funktionen ausgestattet, die die Vorzüge des Macintosh verstärkt nutzbar machen. Das Programm läuft auf allen Macintosh-Computern mit einer Hauptspeicher-Kapazität von mindestens 512 Kbyte. Die neuen Funktionen umfassen:

- PageView, eine WYSIWYG-Editierumgebung,
- Tabellen zur einfachen Positionierung von spaltenbezogenen Absätzen, Zahlen oder Grafiken,
- eine automatische Umpaginierung,
- die Möglichkeit, Text um Objekte herumfließen zu lassen,
- Farb-Unterstützung sowie
- eine automatische Dateien-Verknüpfung mit Microsoft Excel und Microsoft Mail.

Jedes Microsoft Word-4.0-Paket beinhaltet das Word Finder-Wörterbuch und das AutoMac-Makroprogamm.

Neue Funktionen bieten neue Möglichkeiten

Microsoft Word 4.0 bietet nun vier verschiedene Editier-Betriebsarten:

- einen verbesserten »Outlining«-Modus zur Strukturierung des Dokumenteninhalts,
- einen »Galley«-Modus zum schnellen Editieren langer Dokumente,
- einen »Page«-Modus für die seitenorientierte WYSIWYG-Editierung sowie
- einen »Print-Preview«-Modus zur Seitengestaltung.

Mitteilungen . . . . Mitteilungen . . . . Mitteilungen

Mit Microsoft Word 4.0 können Anwender Befehle zu jedem Menü neu zuordnen und die Tastaturbelegung von Befehlen ändern, auch wenn diese Tastaturbefehle nicht in Menüs oder Dialog-Fenstern erscheinen. Der Anwender kann hierfür einfach in einer Liste blättern, die er im Edit-Menü findet. Daraus läßt sich jeder Befehl auswählen und einer bestimmten Tastenkombination oder einem Menü zuordnen.

Neben der größeren Flexibilität, die Microsoft Word 4.0 bietet, verfügt das Programm auch über neue Funktionen, die die Textverarbeitung schneller und einfacher als bisher machen. Eine neue Tabellenfunktion erzeugt beispielsweise ein einstellbares Liniennetz, mit dessen Hilfe spaltenorientierte Absätze, Daten in Form von Kalkulationstabellen sowie Grafiken und Texte einfach positioniert werden können, ohne Tabulatoren setzen zu müssen. Zur Gestaltung und Hervorhebung lassen sich außerdem Rahmen, Unterstreichungen und Linien einsetzen.

Die Fixierung von Objekten an definierten Stellen erleichtert die Erstellung eines Seiten-Layouts. Mit der Absolut-Positionierfunktion können Anwender Grafiken oder Teile des Textes an definierten Stellen auf einer Seite verankern. Zusätzlicher Text fließt dann um die fixierten Objekte herum, ohne sie zu ändern, auch wenn im Text selbst Korrekturen vorgenommen werden.

Microsoft Word 4.0 verfügt über neun »hot spot«-Bereiche auf dem Bildschirm, in denen der Anwender durch ein Doppelklicken der Maus Funktionen aufrufen kann. Erfahrene Anwender sind damit in der Lage, Seiten durchzublättern, Fußnoten einzufügen, Tabulatoren zu verändern, Dokumentfenster zu teilen und anderes mehr. Solche direkten Manipulations-Möglichkeiten über das grafische Bediener-Interface des Macintosh' sind Beispiele dafür, in welcher Weise Microsoft Word 4.0 die Möglichkeiten des Macintosh nutzbar macht.

Mit Hilfe des AutoMac-III-Makroprogramms, das Bestandteil von Microsoft Word 4.0 ist, können Anwender jede Sequenz von häufig verwendeten Befehlen automatisieren. Darüber hinaus bietet die neue Version ein Wörterbuch mit 220.000 Wörtern sowie eine integrierte Wörter-Zählfunktion.

Verbesserte Funktionen

Als Reaktion auf den Wunsch zahlreicher Word-Anwender unterstützt die Version 4.0 nun sowohl eine automatische wie eine Batch-Repaginierung. Anwender, die auf schnellstmögliche Verarbeitung Wert legen, werden Word im Batch-Repaginierungs-Modus einsetzen. Die Automatik-Repaginierungs-Option ist dagegen für diejenigen hilfreich, die zu jeder Zeit den aktuellen Seitenumbruch sehen möchten. Im Seitenbetrachtungs-Modus (PageView Mode) stellt Microsoft Word 4.0 nun editierbare Spalten auf dem Bildschirm dar.

Eine höhere Geschwindigkeit bei einer geringeren Anzahl sichtbarer Seitenelemente auf dem Bildschirm ist das wesentliche Merkmal des »Galley«-Modus (Galley View). Um die Entwicklung und Anwendung von Gestaltungsmustern (style sheets) zu verbessern und zu erleichtern, bietet Microsoft Word 4.0 Gestaltungsbeispiele (style by example). Dabei werden im »Outline view« alle Formatierungen auf dem Bildschirm dargestellt. Inhaltsverzeichnisse und Indextabellen lassen sich durch die Markierung der entsprechenden Textteile automatisch erstellen, so daß auch diese Arbeit mit Microsoft Word 4.0 erleichtert wird.

Das Zusammenspiel mit anderen Anwendungen

Da die meisten Macintosh-Anwender mehrere Software-Pakete einsetzen, hat Microsoft vor allem auch der Integrationsfähigkeit von Word 4.0 große Aufmerksamkeit geschenkt. Die »Word QuickSwitch«-Funktion nutzt das Macintosh-Betriebssystem, zum Aufruf und Laden einer zweiten Anwendung (derzeit Microsoft Excel), wobei dann eine automatische Dateiverbindung hergestellt und aufrechterhalten wird. So ist es beispielsweise möglich, grafische Darstellungen oder Tabellen, die mit Microsoft Excel erstellt wurden, in ein Textdokument zu laden, um sie dort auf einfache Weise zu ergänzen und zu aktualisieren.

Integrale Bestandteile der Microsoft Word-Umgebungen sind Microsoft Mail, Microsoft File und Microsoft PowerPoint. Auf Microsoft Mail kann von Word 4.0 aus zugegriffen werden, um Dokumente, die mit allen verfügbaren »style sheets« formatierbar sind, zu empfangen oder zu senden. Zusammen mit Microsoft File eingesetzt, ist Mailmerge erheblich einfacher anzuwenden, weil Word alle Mailmerge-Felder in einer File-Datenbank anzeigt. Word und PowerPoint arbeiten mit demselben Format für vom Anwender erstellte Wörterbücher, so daß Wörterbücher mit speziellen Begriffen sowohl für die Erstellung von Texten als auch für Präsentations-Dokumente genutzt werden können.

Microsoft Word 4.0 für den Macintosh setzt als Systembasis einen Macintosh Computer mit 512 Kbyte RAM und zwei 800 Kbyte Floppy-Disk Laufwerken oder eine Festplatte voraus. Das Programm ist kompatibel mit AppleShare und MultiFinder.

Microsoft Word 4.0 für den Macintosh ist in der deutschen Version voraussichtlich Ende März 1989 verfügbar und kostet ca. 1.268 DM (zzgl. MwSt.). Die englische Version wird bereits Anfang März 1989 zu einem Preis von ca. 936 DM (zzgl. MwSt.) im Handel sein.

Mitteilungen .... Mitteilungen .... Mitteilungen

Programmiertools für die Benutzeroberfläche

Aspen Scientific (USA) hat die Firma Kickstein Software (Augsburg) mit dem deutschen Exklusiv-Vertrieb ihrer Programmierertools beauftragt. Der Schwerpunkt des Sortiments liegt auf Tools für die Entwicklung von portablen Benutzeroberflächen zwischen MS-DOS, OS/2 und UNIX.

Die Produktpalette umfaßt Tools wie Curses für MS-DOS und OS/2, Formation für MS-DOS, OS/2 und UNIX. Mit diesen Tools sind Programmentwickler in der Lage, ohne großen Aufwand moderne und ausgereifte Benutzeroberflächen zu entwickeln, und diese zwischen MS-DOS, OS/2, UNIX oder XENIX zu portieren. Beide Produkte werden in den USA schon seit Jahren erfolgreich von namhaften Firmen eingesetzt. Kickstein Software unterstützt die Aspen Scientific Produktpalette durch technischen Support und deutsche Dokumentation.

Curses, der Fenstermanager aus der Unix-Welt jetzt unter MS-DOS und OS/2

Aspen Scientifics Curses ist eine komplette Portierung von Curses unter Unix System V.3 auf MS-DOS oder OS/2. Da der neueste Curses-Standard abwärtskompatibel mit älteren Curses-Versionen ist, kann man mit Apsen Scientifics Curses Programme sowohl von Berkely UNIX 4.2 und 4.3 BSD als auch von XENIX portieren.

Curses ist das Unix-Programmierwerkzeug zur Entwicklung von Benutzeroberflächen. Mit dem Curses-Paket können C-Programmierer nun Programme, die Curses verwenden, von Unix auf MS-DOS oder umgekehrt übertragen. Gleichermaßen steht dem Entwickler der komplette mächtige Befehlsumfang von Curses für Entwicklungen in einer MS-DOS Umgebung zur Verfügung. Curses arbeitet geräteunabhängig, d.h. ein Entwickler braucht ein Programm nur einmal zu schreiben um es dann unter allen MS-DOS-Bildschirmschnittstellen laufen zu lassen.

Curses unterstützt derzeit folgende Adapter: EGA (inkl. 43-Zeichen-Modus), CGA, MDA, BIOS Farb- und Monochromanzeige sowie ANSI-Bildschirmsteuerung in Farbe und monochrom. Curses ist sehr schnell, denn es arbeitet mit derselben Bildschirmoptimierung wie Unix-Curses.

Im Lieferumfang von Curses sind zusätzlich die Bildschirmmaskenverwaltung FAST (mit Quelltexten), das Online-Hilfesystem QuickHELP und der interaktive C-Quellencodegenerator, Editor und Interpreter TUTOR enthalten. Mit TUTOR schreibt man sein erstes Curses-Programm bereits nach wenigen Minuten.

Das Handbuch von Curses ist zur Zeit noch in englisch, ein deutsches Handbuch ist in Arbeit. Curses ist für alle gängigen C-Compiler, z.B. auch Microsoft C, erhältlich.

Curses ist ab sofort als Binärpaket (Funktionsbibliotheken) mit TUTOR, FAST und QuickHELP zum empfohlenen Verkaufspreis von DM 298,- inkl. MwSt. oder mit allen Curses-Quelltexten zum empfohlenen Verkaufspreis von DM 720,- inkl. MwSt. lieferbar.

Formation, der Fenster-, Menü- und Dialogboxen-Manger unter Curses für MS-DOS, OS/2 und UNIX

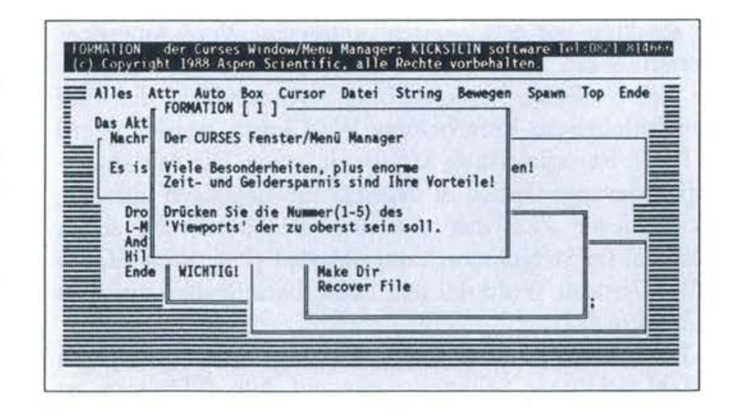
Mit Aspen Scientifics Formation steht dem Softwareentwickler ein mächtiges Werkzeug für eine moderne Benutzeroberfläche zur Verfügung. Es eröffnet ihm im Textmodus Möglichkeiten, die er sonst nur in einer grafischen Umgebung hat. Anders als bei herkömmlichen C-Fensterbibliotheken, unterstützt Formation sogenannte »Viewports«, große virtuelle Fenster (sogenannte »Pads«), Pop-Up-, Drop-Down- und Balken-Menüs, Dialogboxen und vieles andere mehr. In einem Formation »Viewport« kann man eine Menge verschiedenster Operationen wie Scrollen, Einfügen und Löschen von Zeilen und Zeichen, Attribute ändern, Zeichen schreiben und wiederholen, Linien und Boxen zeichnen und vieles mehr durchführen. Formation verwaltet automatisch die Wiederherstellung von sich überschneidenden »Viewports«.

Dem Entwickler stehen drei verschiedene Menüschnittstellen zur Verfügung: Pop-Up-, Drop-Down- und Balken-Menüs. Mit ihren Dialogboxen ist Aspen Scientific einen anderen Weg als die vielen zeichenorientierten Fenstertools gegangen. Sie besitzen mehrere Arten von Eingabemedien: scrollende Eingabefelder, Zusatzknöpfe, Checkboxen, scrollende Übersichtsboxen und Befehlsknöpfe.

Zusätzlich ist im Lieferumfang von Formation noch das Online-Hilfesystem QuickHELP enthalten. Das Handbuch ist zur Zeit noch in englisch, eine deutsches Handbuch ist jedoch bereits in Arbeit. Formation ist für alle gängigen C-Compiler, zum Beispiel auch Microsoft C, erhältlich.

Um eine 100%ige Portabilität zu erreichen, baut Formation auf Aspen Scientifics Curses auf. So kann ein Programmierer jederzeit seine Anwendungen auf alle Systeme portieren, welche die Curses-Schnittstelle unterstützen (wie z.B. UNIX, DOS und OS/2).

Formation ist ab sofort als Binärpaket (Funktionsbibliotheken) mit QuickHELP zum empfohlenen Verkaufspreis von DM 398,- inkl. MwSt. oder mit allen Formation-Quelltexten zum empfohlenen Verkaufspreis von DM 748,- inkl. MwSt. lieferbar. Eine Demodiskette ist für einen Unkostenbeitrag von DM 10,- bei Kickstein Software, Augsburg, erhältlich.



Kurzkritik .... Kurzkritik .... Kurzkritik

Von 3,5" auf 5,25" und umgekehrt

Das Problem ist bekannt: Der AT, mit dem man arbeitet, kann keine 3,5-Zoll-Disketten lesen bzw. das PS/2-Modell keine 5,25-Zoll-Disketten. Abhilfe schaffen hier die beiden kleinen »Kisten« von Sysgen, genannt Bridge-File, die in Deutschland von M+S Elektronik vertrieben werden.

Das externe Bridge-File gibt es in der 5,25-Zoll-Version (360 Kbyte und 1,2 Mbyte) für die PS/2-Modelle 30, 50/50Z, 60/80 und 70 sowie in der 3,5-Zoll-Version (720 Kbyte und 1,44 Mbyte) für die PC-Modelle bis zum AT. Beide kosten einschließlich Adapter und Mehrwertsteuer 1345,20 DM.

Die Installation des 3,5-Zoll-Laufwerks, das uns zum Test zur Verfügung stand, war denkbar einfach. Eine kurze Karte wird in die Zentraleinheit gesteckt und über ein mitgeliefertes Kabel zwischen den bereits vorhandenen Disk-Controller und das eingebaute 5,25-Zoll-Laufwerk geschleift. Die Verbindung zum externen Bridge-File wird über ein dickes Kabel mit zwei gleichen 37-poligen Steckern hergestellt.

Nun muß noch die Software-Installation vorgenommen werden. Auf der beigelegten 5,25-Zoll-Diskette befinden sich dazu ein Install-Programm und zwei Treiber. Nach Aufruf dieses Programms wird zunächst einmal der benutzte Computer ausgewählt (*Bild 1*). Sind dann auch die übrigen Fragen beantwortet, werden die Angaben zur Überprüfung angezeigt (*Bild 2*). Durch die Bestätigung mit »Y« kopiert das Installationsprogramm den richtigen Treiber auf das gewünschte Laufwerk und trägt in die Konfigurationsdatei diesen Einheitentreiber ein.

Jetzt kommen die beiden Utilities zur Anwendung. Mit `VIEW` kann man sich den Laufwerksbuchstaben anzeigen lassen, mit dem das neue Laufwerk anzusprechen ist. Die zweite Utility, `FORMAT35`, dient zum Vorbereiten der Disketten. Abhängig vom Schalter `/F:3` oder `/F:4` wird die 3,5-Zoll-Diskette mit 720 Kbyte bzw. 1,44 Mbyte formatiert.

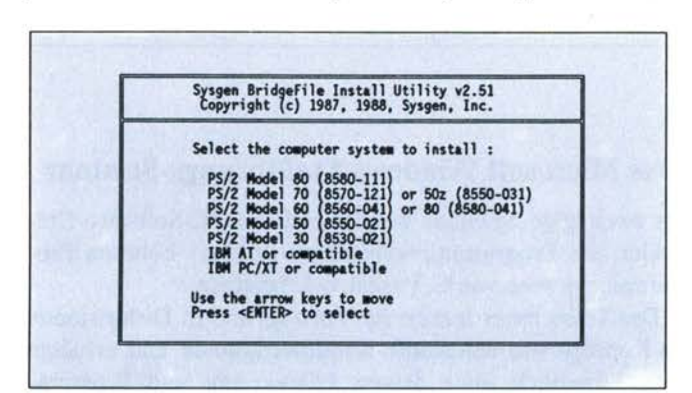


Bild 1: *Die Installation beginnt mit der Auswahl des Computermodells.*

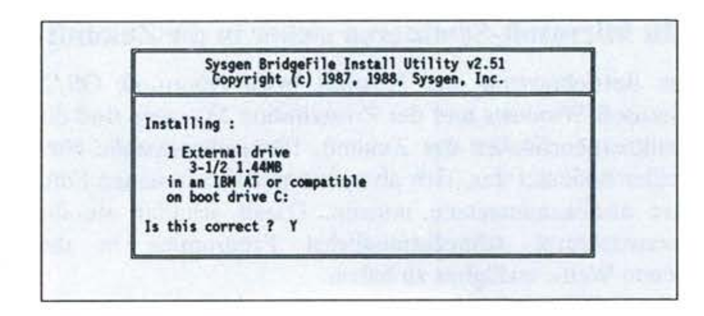


Bild 2: *Zur Kontrolle werden noch einmal die ausgewählten Daten angezeigt.*

Fazit

Weder das Laufwerk, noch die Installation machten irgendwelche Probleme. Nicht zuletzt aufgrund des ausführlichen, jedoch englischen Handbuchs. Schade ist eigentlich nur, daß Bridge-File keine Macintosh-Disketten verarbeiten kann.

ni

Buchkatalog auf Diskette

Einen Elektronik-Fachbuch-Katalog als Datenbank auf Diskette bietet die Firma Feltron-Zeissler an. Diese mit Clipper realisierte Anwendung bietet auf inzwischen vier 360-Kbyte-Disketten über 4200 Titel der wichtigsten deutschen Verlage zur Auswahl an.

Die Daten können entweder nach der Bestellnummer, oder einem Stichwort durchsucht werden. Wird z.B das Stichwort »Word« gewählt, erscheinen nacheinander die Word-Titel mit den in *Bild 3* dargestellten Informationen.

Eine solche Buch-Disk ist fast kostenlos. So werden die ersten drei Disketten für unglaubliche 7,20 DM inklusive Versandkosten angeboten. Natürlich macht Feltron-Zeissler das nicht ganz uneigennützig und bietet gleich die schnelle Auslieferung der gewünschten Bücher innerhalb eines Tages an. Ein Service also, der gerade auf dem »flachen Lande« nicht zu unterschätzen ist.

ni



Bild 3: *Bei der Suche nach »Word« erscheinen außer Word-Büchern auch Titel zu Wordstar oder Word Perfect.*

Termine ... Termine ... Termine ... Termine

Mit Microsoft-Seminaren sicher in die Zukunft

Das Betriebssystem der Zukunft heißt Microsoft OS/2. Microsoft Windows und der Presentation Manager sind die Benutzeroberflächen der Zukunft. Für professionelle Entwickler bedeutet das, sich ab sofort mit dieser neuen Software auseinandersetzen müssen. Damit schaffen sie die Voraussetzung, schnellstmöglichst Programme in der »neuen Welt« verfügbar zu haben.

Natürlich wird die Umstellung auf das neue Betriebssystem sowie die Programmerstellung nicht von heute auf morgen vollzogen sein. Um den Anfang jedoch so einfach wie möglich zu gestalten, bietet Microsoft eine Dienstleistung an: Das Microsoft Institut.

Die Spezialseminare des Microsoft Instituts vermitteln in kleinen Gruppen intensiv all das, was zum Einstieg in die Programmentwicklung nötig ist. Modernste Trainingsmethoden sowie PC-Demonstrationen und -Übungen sind selbstverständlich. Die Dozenten befassen sich auch im persönlichen Gespräch ausführlich mit den individuellen Forderungen und Problemen der Teilnehmer. So bekommen professionelle Entwickler durch professionelle Schulung die Möglichkeit, ihren hohen Wissenstand den neuen Gegebenheiten anzupassen.

Jeder Interessent in der Bundesrepublik Deutschland, der Schweiz und Österreich hat die Chance, sich mit der neuen Welt von Microsoft OS/2 und Microsoft Windows auseinanderzusetzen. Denn das Microsoft Institut arbeitet vor Ort mit kompetenten Schulungsunternehmen zusammen:

Digicomp AG, Zürich
Elektro-Calcul PI S.A., Lausanne
Integrata GmbH, Tübingen
INTEL Semiconductor GmbH, Feldkirchen/München
Olivetti Bildungs-Zentrum GmbH, Düsseldorf
Ueberreuter Media GmbH, Wien.

Die Dozenten werden speziell von Microsoft ausgebildet und stehen in ständigem Kontakt mit uns. So gibt es keine Informationsverluste: Das Wissen wird immer aktuell und aus erster Hand vermittelt. Microsoft erstellt die Seminare und die Seminarunterlagen und gewährleistet Qualität durch die Auswertung der Seminare.

Das Microsoft OS/2 Einführungs-Seminar

Das zweitägige Seminar wendet sich an PC-Software-Entwickler, die Programmierkenntnisse in einer höheren Programmiersprache wie C, Pascal, o.ä. besitzen.

Die Teilnehmer lernen im Vortrag und in Diskussionen das Konzept von Microsoft OS/2 kennen und erhalten einen Überblick über die Fähigkeiten und Programmierschnittstellen dieses Betriebssystems. Während des Seminars haben die Teilnehmer die Möglichkeit, das Gelernte anhand von Übungsaufgaben für sich selbst zu überprüfen.

Ort	Datum	Veranstalter
Düsseldorf	16./17.01.	OBZ
	13./14.02.	OBZ
	06./07.03.	OBZ
Graz	15./16.03.	Ueberreuter
Hamburg	06./07.02.	Integrata
	09./10.03.	Integrata
Innsbruck	12./13.01.	Ueberreuter
Lausanne	13./14.02.	Electro Calcul
Linz	23./24.01.	Ueberreuter
Salzburg	13./14.03.	Ueberreuter
Wien	20./21.02.	Ueberreuter
Zürich	30./31.01.	Digicomp

Der Microsoft OS/2 Workshop

Das dreitägige Seminar wendet sich an PC-Software-Entwickler, die Programmiererfahrungen in einer höheren, strukturierten Programmiersprache unter MS-DOS und C-Kenntnisse besitzen sowie das MS-OS/2-Einführungsseminar besucht haben.

Die Teilnehmer lernen im Vortrag und praktischen Übungen am PC Family-API-Programme zu schreiben und Device-I/O-Routinen zu erstellen sowie Multitasking-Funktionen zu nutzen und eigene Dynamic-Link-Bibliotheken zu erstellen; außerdem können sie die erweiterten Speicherverwaltungsmöglichkeiten des Intel 80286 nutzen und mit Hilfe des MS-OS/2 Memory Managers programmieren. Dieses Seminar ist übrigens nicht im SDK-Preis enthalten.

Ort	Datum	Veranstalter
Düsseldorf	18./19./20.01.	OBZ
	15./16./17.02.	OBZ
	08./09./10.03.	OBZ
Frankfurt	25./26./27.01.	Integrata
Hamburg	08./09./10.02.	Integrata
Lausanne	22./23./24.02.	Electro Calcul
Tübingen	29./30./31.03.	Integrata
Wien	11./12./13.01.	Ueberreuter
	01./02./03.03.	Ueberreuter
Zürich	06./07./13.03.	Digicomp

Das Microsoft Windows Einführungs-Seminar

Das zweitägige Seminar wendet sich an PC-Software-Entwickler, die Programmierkenntnisse in einer höheren Programmiersprache wie C, Pascal, o.ä. besitzen.

Die Teilnehmer lernen im Vortrag und in Diskussionen das Konzept von Microsoft Windows kennen und erhalten einen Überblick über dessen Fähigkeiten und Programmierschnittstellen. Dieses Seminar ist nicht im SDK-Preis enthalten.

Termine ... Termine ... Termine ... Termine

Ort	Datum	Veranstalter
Düsseldorf	23./24.01.	OBZ
	20./21.02.	OBZ
	13./14.03.	OBZ
Frankfurt	06./07.03.	Integrata
Innsbruck	06./07.02.	Ueberreuter
Lausanne	06./07.03.	Electro Calcul
München	13./14.02.	Integrata
Salzburg	13./14.03.	Ueberreuter
Wien	26./27.01.	Ueberreuter
Zürich	auf Anfrage	Digicomp

Der Microsoft Windows Workshop

Das dreitägige Seminar wendet sich an PC-Software-Entwickler, die Programmiererfahrungen in einer höheren, strukturierten Programmiersprache unter MS-DOS und C-Kenntnisse besitzen sowie das Microsoft Windows Einführungsseminar besucht haben.

Die Teilnehmer lernen im Vortrag und praktischen Übungen am PC Benutzerschnittstellen zu erstellen, die grafische Programmierschnittstelle zu nutzen, die Routinen zum Memory Management anzuwenden und dynamische Bibliotheken zu erstellen und zu benutzen. Dieses Seminar ist nicht im SDK-Preis enthalten.

Ort	Datum	Veranstalter
Düsseldorf	25./26./27.01.	OBZ
	22./23./24.02.	OBZ
	15./16./17.03.	OBZ
Frankfurt	08./09./10.03.	Integrata
Lausanne	08./09./10.03	Electro Calcul
München	15./16./17.02.	Integrata
Wien	22./23./24.02.	Ueberreuter
Zürich	auf Anfrage	Digicomp

Microsoft Excel für Programmierer

Das dreitägige Seminar wendet sich an PC-Software-Entwickler, die Programmiererfahrung in einer höheren Programmiersprache haben.

Die Teilnehmer lernen im Vortrag und in praktischen Übungen am PC das Konzept und die Möglichkeiten, mit Microsoft Excel-Makros Applikationen zu erstellen.

Ort	Datum	Veranstalter
Düsseldorf	30./31.01./01.02.	OBZ
	27./28.02./01.03.	OBZ
	20./21./22.03.	OBZ
Genf	05./06./07.12.	Electro Calcul
Hamburg	20./21./22.02.	Integrata
Tübingen	16./17./18.01.	Integrata
Wien	01./02./03.02.	Ueberreuter
	13./14./15.03.	Ueberreuter

Microsoft Presentation Manager für Windows Programmierer

Dieses Seminar erleichtert das Umsteigen von Microsoft Windows auf den Microsoft OS/2 Presentation Manager.

Der Teilnehmer lernt die Unterschiede zwischen Microsoft Windows und dem Microsoft OS/2 Presentation Manager kennen und lernt, wie er seine Windows-Applikationen auf den PM übertragen kann.

Ort	Datum	Veranstalter
Düsseldorf	30.01.89	OBZ
	02.03.89	OBZ
Frankfurt	31.01.89	Integrata
Lausanne	14.03.89	Electro Calcul
München	20.02.89	Integrata
Tübingen	17.03.89	Integrata
Wien	25.01.89	Ueberreuter

Inserentenverzeichnis

Allgemeingültige Zeichenwandlung und Klassifizierung:

Nützliche Zeichenroutinen in Assembler

In fast allen Programmen benötigt man Routinen, die Zeichenvergleiche oder die Einordnung von Zeichen in Zeichenklassen ermöglichen (alphanumerisch, Ziffer usw.). In den meisten Programmiersprachen sind solche Routinen enthalten, z.B. isupper(), tolower() in C. Der Assemblerprogrammierer muß sich so etwas selbst bauen. Hier wird eine allgemeingültige Lösung dafür vorgestellt.

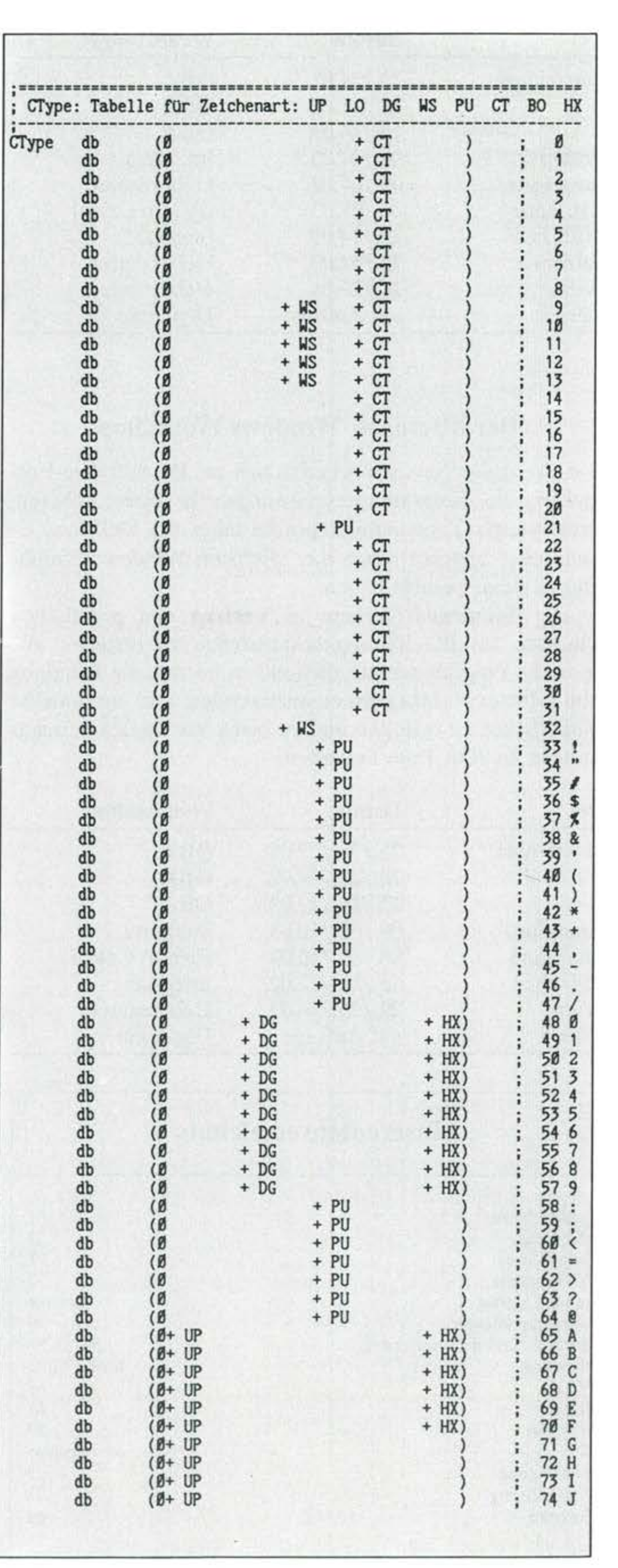
Das Herzstück der gezeigten Routinen sind drei Tabellen:
CType enthält Informationen über die Zeichenklasse, z.B. Klein- oder Großbuchstabe, Ziffer usw.
CConv enthält für Großbuchstaben an der entsprechenden Zeichenposition die Kleinbuchstaben und umgekehrt. Diese Tabelle wird für Groß-/Kleinwandlungen benötigt.
CLex enthält für jedes Zeichen seinen lexikalischen Wert. Damit können Zeichen unabhängig von ihrer Position im Zeichensatz lexikalisch richtig sortiert werden.

Das Programmodul IS (*Listing 1*) enthält folgende Beispielroutinen:
IsDigit prüft, ob das Zeichen in AL eine Ziffer ist. Handelt es sich um eine Ziffer, wird das Carry-Flag gesetzt. `IsDigit` führt den Vergleich noch auf traditionelle Weise direkt durch. Das ist bei Ziffern kein Problem. Besser wäre hier jedoch eine Vorgehensweise, wie sie in der folgenden Routine zu sehen ist.
IsAlpha prüft, ob das Zeichen in AL aphanumerisch ist (Groß- oder Kleinbuchstabe). Ist das Zeichen alphanumerisch, wird das Carry-Flag gesetzt. `IsAlpha` geht zur Bestimmung der Zeichenklasse über die Tabelle `CType`.
ToUpper wandelt das Zeichen in AL in einen Großbuchstaben, wenn es sich um einen Kleinbuchstaben handelt. `ToUpper` geht zur Feststellung der Zeichenklasse über die Tabelle `CType` und wandelt dann das Zeichen über `CConv`.
ToLex erhält ein Zeichen in AL und gibt seinen Sortierwert aus der Tabelle `CLex` in AL zurück.

In der Art dieser Routinen kann man nun sehr einfach weitere Zeichenwandlungs- und klassifizierungroutinen schreiben, wie zum Beispiel `IsUpper`, `ToLower` usw.

```
;===============================================================
; Modul IS
; (C) 1989 Günter Jürgensmeier
;===============================================================
        .DATA
;===============================================================
UP      equ     01h     ; uppercase     Großbuchstabe
LO      equ     02h     ; lowercase     Kleinbuchstabe
DG      equ     04h     ; digit         Ziffer
WS      equ     08h     ; white space   Leerzeichen, Tab usw.
PU      equ     10h     ; punctuation   Satzzeichen
CT      equ     20h     ; control       Steuerzeichen
BO      equ     40h     ; box           Liniengrafikzeichen
HX      equ     80h     ; hex digit     hexadezimale Ziffer
```

Listing 1: *Zeichenroutinen in Assembler*



```
;===============================================================
; CType: Tabelle für Zeichenart: UP  LO  DG  WS  PU  CT  BO  HX
;---------------------------------------------------------------
CType   db      (0                         + CT          )   ;  0
        db      (0                         + CT          )   ;  1
        db      (0                         + CT          )   ;  2
        db      (0                         + CT          )   ;  3
        db      (0                         + CT          )   ;  4
        db      (0                         + CT          )   ;  5
        db      (0                         + CT          )   ;  6
        db      (0                         + CT          )   ;  7
        db      (0                         + CT          )   ;  8
        db      (0               + WS      + CT          )   ;  9
        db      (0               + WS      + CT          )   ; 10
        db      (0               + WS      + CT          )   ; 11
        db      (0               + WS      + CT          )   ; 12
        db      (0               + WS      + CT          )   ; 13
        db      (0                         + CT          )   ; 14
        db      (0                         + CT          )   ; 15
        db      (0                         + CT          )   ; 16
        db      (0                         + CT          )   ; 17
        db      (0                         + CT          )   ; 18
        db      (0                         + CT          )   ; 19
        db      (0                         + CT          )   ; 20
        db      (0                    + PU               )   ; 21
        db      (0                         + CT          )   ; 22
        db      (0                         + CT          )   ; 23
        db      (0                         + CT          )   ; 24
        db      (0                         + CT          )   ; 25
        db      (0                         + CT          )   ; 26
        db      (0                         + CT          )   ; 27
        db      (0                         + CT          )   ; 28
        db      (0                         + CT          )   ; 29
        db      (0                         + CT          )   ; 30
        db      (0                         + CT          )   ; 31
        db      (0               + WS                    )   ; 32
        db      (0                    + PU               )   ; 33 !
        db      (0                    + PU               )   ; 34 "
        db      (0                    + PU               )   ; 35 #
        db      (0                    + PU               )   ; 36 $
        db      (0                    + PU               )   ; 37 %
        db      (0                    + PU               )   ; 38 &
        db      (0                    + PU               )   ; 39 '
        db      (0                    + PU               )   ; 40 (
        db      (0                    + PU               )   ; 41 )
        db      (0                    + PU               )   ; 42 *
        db      (0                    + PU               )   ; 43 +
        db      (0                    + PU               )   ; 44 ,
        db      (0                    + PU               )   ; 45 -
        db      (0                    + PU               )   ; 46 .
        db      (0                    + PU               )   ; 47 /
        db      (0          + DG                     + HX)   ; 48 0
        db      (0          + DG                     + HX)   ; 49 1
        db      (0          + DG                     + HX)   ; 50 2
        db      (0          + DG                     + HX)   ; 51 3
        db      (0          + DG                     + HX)   ; 52 4
        db      (0          + DG                     + HX)   ; 53 5
        db      (0          + DG                     + HX)   ; 54 6
        db      (0          + DG                     + HX)   ; 55 7
        db      (0          + DG                     + HX)   ; 56 8
        db      (0          + DG                     + HX)   ; 57 9
        db      (0                    + PU               )   ; 58 :
        db      (0                    + PU               )   ; 59 ;
        db      (0                    + PU               )   ; 60 <
        db      (0                    + PU               )   ; 61 =
        db      (0                    + PU               )   ; 62 >
        db      (0                    + PU               )   ; 63 ?
        db      (0                    + PU               )   ; 64 @
        db      (0+ UP                               + HX)   ; 65 A
        db      (0+ UP                               + HX)   ; 66 B
        db      (0+ UP                               + HX)   ; 67 C
        db      (0+ UP                               + HX)   ; 68 D
        db      (0+ UP                               + HX)   ; 69 E
        db      (0+ UP                               + HX)   ; 70 F
        db      (0+ UP                                   )   ; 71 G
        db      (0+ UP                                   )   ; 72 H
        db      (0+ UP                                   )   ; 73 I
        db      (0+ UP                                   )   ; 74 J
```

Listing 1: *(Fortsetzung)*

```
db      (Ø+ UP                              )   ;  75 K
db      (Ø+ UP                              )   ;  76 L
db      (Ø+ UP                              )   ;  77 M
db      (Ø+ UP                              )   ;  78 N
db      (Ø+ UP                              )   ;  79 O
db      (Ø+ UP                              )   ;  8Ø P
db      (Ø+ UP                              )   ;  81 Q
db      (Ø+ UP                              )   ;  82 R
db      (Ø+ UP                              )   ;  83 S
db      (Ø+ UP                              )   ;  84 T
db      (Ø+ UP                              )   ;  85 U
db      (Ø+ UP                              )   ;  86 V
db      (Ø+ UP                              )   ;  87 W
db      (Ø+ UP                              )   ;  88 X
db      (Ø+ UP                              )   ;  89 Y
db      (Ø+ UP                              )   ;  9Ø Z
db      (Ø                + PU              )   ;  91 [
db      (Ø                + PU              )   ;  92 \
db      (Ø                + PU              )   ;  93 ]
db      (Ø                + PU              )   ;  94 ^
db      (Ø                + PU              )   ;  95
db      (Ø                + PU              )   ;  96 '
db      (Ø    + LO                    + HX) ;  97 a
db      (Ø    + LO                    + HX) ;  98 b
db      (Ø    + LO                    + HX) ;  99 c
db      (Ø    + LO                    + HX) ; 1ØØ d
db      (Ø    + LO                    + HX) ; 1Ø1 e
db      (Ø    + LO                    + HX) ; 1Ø2 f
db      (Ø    + LO                          )   ; 1Ø3 g
db      (Ø    + LO                          )   ; 1Ø4 h
db      (Ø    + LO                          )   ; 1Ø5 i
db      (Ø    + LO                          )   ; 1Ø6 j
db      (Ø    + LO                          )   ; 1Ø7 k
db      (Ø    + LO                          )   ; 1Ø8 l
db      (Ø    + LO                          )   ; 1Ø9 m
db      (Ø    + LO                          )   ; 11Ø n
db      (Ø    + LO                          )   ; 111 o
db      (Ø    + LO                          )   ; 112 p
db      (Ø    + LO                          )   ; 113 q
db      (Ø    + LO                          )   ; 114 r
db      (Ø    + LO                          )   ; 115 s
db      (Ø    + LO                          )   ; 116 t
db      (Ø    + LO                          )   ; 117 u
db      (Ø    + LO                          )   ; 118 v
db      (Ø    + LO                          )   ; 119 w
db      (Ø    + LO                          )   ; 12Ø x
db      (Ø    + LO                          )   ; 121 y
db      (Ø    + LO                          )   ; 122 z
db      (Ø                + PU              )   ; 123 {
db      (Ø                + PU              )   ; 124 +
db      (Ø                + PU              )   ; 125 }
db      (Ø                + PU              )   ; 126 ~
db      (Ø                    + CT          )   ; 127
db      (Ø+ UP                              )   ; 128
db      (Ø    + LO                          )   ; 129 ü
db      (Ø    + LO                          )   ; 13Ø
db      (Ø    + LO                          )   ; 131
db      (Ø    + LO                          )   ; 132 ä
db      (Ø    + LO                          )   ; 133
db      (Ø    + LO                          )   ; 134
db      (Ø    + LO                          )   ; 135
db      (Ø    + LO                          )   ; 136
db      (Ø    + LO                          )   ; 137
db      (Ø    + LO                          )   ; 138
db      (Ø    + LO                          )   ; 139
db      (Ø    + LO                          )   ; 14Ø
db      (Ø    + LO                          )   ; 141
db      (Ø+ UP                              )   ; 142 Ä
db      (Ø+ UP                              )   ; 143
db      (Ø+ UP                              )   ; 144
db      (Ø    + LO                          )   ; 145
db      (Ø+ UP                              )   ; 146
db      (Ø    + LO                          )   ; 147
db      (Ø    + LO                          )   ; 148 ö
db      (Ø    + LO                          )   ; 149
db      (Ø    + LO                          )   ; 15Ø
db      (Ø    + LO                          )   ; 151
db      (Ø    + LO                          )   ; 152
```

Listing 1: *(Fortsetzung)*

```
db      (Ø+ UP                              )   ; 153 Ö
db      (Ø+ UP                              )   ; 154 Ü
db      (Ø                + PU              )   ; 155
db      (Ø                + PU              )   ; 156
db      (Ø                + PU              )   ; 157
db      (Ø                + PU              )   ; 158
db      (Ø                + PU              )   ; 159
db      (Ø    + LO                          )   ; 16Ø
db      (Ø    + LO                          )   ; 161
db      (Ø    + LO                          )   ; 162
db      (Ø    + LO                          )   ; 163
db      (Ø    + LO                          )   ; 164
db      (Ø+ UP                              )   ; 165
db      (Ø                + PU              )   ; 166
db      (Ø                + PU              )   ; 167
db      (Ø                + PU              )   ; 168
db      (Ø                + PU              )   ; 169 ⌐
db      (Ø                + PU              )   ; 17Ø ¬
db      (Ø                + PU              )   ; 171 ½
db      (Ø                + PU              )   ; 172 ¼
db      (Ø                + PU              )   ; 173
db      (Ø                + PU              )   ; 174 «
db      (Ø                + PU              )   ; 175 »
db      (Ø                        + BO      )   ; 176 ░
db      (Ø                        + BO      )   ; 177 ▒
db      (Ø                        + BO      )   ; 178 ▓
db      (Ø                        + BO      )   ; 179 │
db      (Ø                        + BO      )   ; 18Ø ┤
db      (Ø                        + BO      )   ; 181 ╡
db      (Ø                        + BO      )   ; 182 ╢
db      (Ø                        + BO      )   ; 183 ╖
db      (Ø                        + BO      )   ; 184 ╕
db      (Ø                        + BO      )   ; 185 ╣
db      (Ø                        + BO      )   ; 186 ║
db      (Ø                        + BO      )   ; 187 ╗
db      (Ø                        + BO      )   ; 188 ╝
db      (Ø                        + BO      )   ; 189 ╜
db      (Ø                        + BO      )   ; 19Ø ╛
db      (Ø                        + BO      )   ; 191 ┐
db      (Ø                        + BO      )   ; 192 └
db      (Ø                        + BO      )   ; 193 ┴
db      (Ø                        + BO      )   ; 194 ┬
db      (Ø                        + BO      )   ; 195 ├
db      (Ø                        + BO      )   ; 196 ─
db      (Ø                        + BO      )   ; 197 ┼
db      (Ø                        + BO      )   ; 198 ╞
db      (Ø                        + BO      )   ; 199 ╟
db      (Ø                        + BO      )   ; 2ØØ ╚
db      (Ø                        + BO      )   ; 2Ø1 ╔
db      (Ø                        + BO      )   ; 2Ø2 ╩
db      (Ø                        + BO      )   ; 2Ø3 ╦
db      (Ø                        + BO      )   ; 2Ø4 ╠
db      (Ø                        + BO      )   ; 2Ø5 ═
db      (Ø                        + BO      )   ; 2Ø6 ╬
db      (Ø                        + BO      )   ; 2Ø7 ╧
db      (Ø                        + BO      )   ; 2Ø8 ╨
db      (Ø                        + BO      )   ; 2Ø9 ╤
db      (Ø                        + BO      )   ; 21Ø ╥
db      (Ø                        + BO      )   ; 211 ╙
db      (Ø                        + BO      )   ; 212 ╘
db      (Ø                        + BO      )   ; 213 ╒
db      (Ø                        + BO      )   ; 214 ╓
db      (Ø                        + BO      )   ; 215 ╫
db      (Ø                        + BO      )   ; 216 ╪
db      (Ø                        + BO      )   ; 217 ┘
db      (Ø                        + BO      )   ; 218 ┌
db      (Ø                        + BO      )   ; 219 █
db      (Ø                        + BO      )   ; 22Ø ▄
db      (Ø                        + BO      )   ; 221 ▌
db      (Ø                        + BO      )   ; 222 ▐
db      (Ø                        + BO      )   ; 223 ▀
db      (Ø                                  )   ; 224
db      (Ø    + LO                          )   ; 225 ß
db      (Ø                                  )   ; 226
db      (Ø                                  )   ; 227
db      (Ø                                  )   ; 228
db      (Ø                                  )   ; 229
db      (Ø                                  )   ; 23Ø
```

Listing 1: *(Fortsetzung)*

```
        db      (Ø                                  )  ; 231
        db      (Ø                                  )  ; 232 ▼
        db      (Ø                                  )  ; 233
        db      (Ø                                  )  ; 234
        db      (Ø                                  )  ; 235
        db      (Ø                                  )  ; 236
        db      (Ø                                  )  ; 237
        db      (Ø                                  )  ; 238
        db      (Ø                + PU              )  ; 239
        db      (Ø                + PU              )  ; 24Ø
        db      (Ø                + PU              )  ; 241 ±
        db      (Ø                + PU              )  ; 242 ≥
        db      (Ø                + PU              )  ; 243 ≤
        db      (Ø                + PU              )  ; 244
        db      (Ø                + PU              )  ; 245
        db      (Ø                + PU              )  ; 246 ÷
        db      (Ø                + PU              )  ; 247 ≈
        db      (Ø                + PU              )  ; 248 °
        db      (Ø                + PU              )  ; 249 ·
        db      (Ø                + PU              )  ; 25Ø ·
        db      (Ø                + PU              )  ; 251 √
        db      (Ø                + PU              )  ; 252
        db      (Ø                + PU              )  ; 253
        db      (Ø                + PU              )  ; 254 ■
        db      (Ø                                  )  ; 255

;==============================================================
CConv   db      ØØØH,ØØ1H,ØØ2H,ØØ3H,ØØ4H,ØØ5H,ØØ6H,ØØ7H
        db      ØØ8H,ØØ9H,ØØaH,ØØbH,ØØcH,ØØdH,ØØeH,ØØfH
        db      Ø1ØH,Ø11H,Ø12H,Ø13H,Ø14H,Ø15H,Ø16H,Ø17H
        db      Ø18H,Ø19H,Ø1aH,Ø1bH,Ø1cH,Ø1dH,Ø1eH,Ø1fH
        db      Ø2ØH,Ø21H,Ø22H,Ø23H,Ø24H,Ø25H,Ø26H,Ø27H
        db      Ø28H,Ø29H,Ø2aH,Ø2bH,Ø2cH,Ø2dH,Ø2eH,Ø2fH
        db      Ø3ØH,Ø31H,Ø32H,Ø33H,Ø34H,Ø35H,Ø36H,Ø37H
        db      Ø38H,Ø39H,Ø3aH,Ø3bH,Ø3cH,Ø3dH,Ø3eH,Ø3fH
        db      Ø4ØH,Ø61H,Ø62H,Ø63H,Ø64H,Ø65H,Ø66H,Ø67H
        db      Ø68H,Ø69H,Ø6aH,Ø6bH,Ø6cH,Ø6dH,Ø6eH,Ø6fH
        db      Ø7ØH,Ø71H,Ø72H,Ø73H,Ø74H,Ø75H,Ø76H,Ø77H
        db      Ø78H,Ø79H,Ø7aH,Ø5bH,Ø5cH,Ø5dH,Ø5eH,Ø5fH
        db      Ø6ØH,Ø41H,Ø42H,Ø43H,Ø44H,Ø45H,Ø46H,Ø47H
        db      Ø48H,Ø49H,Ø4aH,Ø4bH,Ø4cH,Ø4dH,Ø4eH,Ø4fH
        db      Ø5ØH,Ø51H,Ø52H,Ø53H,Ø54H,Ø55H,Ø56H,Ø57H
        db      Ø58H,Ø59H,Ø5aH,Ø7bH,Ø7cH,Ø7dH,Ø7eH,Ø7fH
        db      Ø87H,Ø9aH,Ø9ØH,Ø83H,Ø8eH,Ø85H,Ø8fH,Ø8ØH
        db      Ø88H,Ø89H,Ø8aH,Ø8bH,Ø8cH,Ø8dH,Ø84H,Ø86H
        db      Ø82H,Ø92H,Ø91H,Ø93H,Ø99H,Ø95H,Ø96H,Ø97H
        db      Ø98H,Ø94H,Ø81H,Ø9bH,Ø9cH,Ø9dH,Ø9eH,Ø9fH
        db      ØaØH,Øa1H,Øa2H,Øa3H,Øa5H,Øa4H,Øa6H,Øa7H
        db      Øa8H,Øa9H,ØaaH,ØabH,ØacH,ØadH,ØaeH,ØafH
        db      ØbØH,Øb1H,Øb2H,Øb3H,Øb4H,Øb5H,Øb6H,Øb7H
        db      Øb8H,Øb9H,ØbaH,ØbbH,ØbcH,ØbdH,ØbeH,ØbfH
        db      ØcØH,Øc1H,Øc2H,Øc3H,Øc4H,Øc5H,Øc6H,Øc7H
        db      Øc8H,Øc9H,ØcaH,ØcbH,ØccH,ØcdH,ØceH,ØcfH
        db      ØdØH,Ød1H,Ød2H,Ød3H,Ød4H,Ød5H,Ød6H,Ød7H
        db      Ød8H,Ød9H,ØdaH,ØdbH,ØdcH,ØddH,ØdeH,ØdfH
        db      ØeØH,Øe1H,Øe2H,Øe3H,Øe4H,Øe5H,Øe6H,Øe7H
        db      Øe8H,Øe9H,ØeaH,ØebH,ØecH,ØedH,ØeeH,ØefH
        db      ØfØH,Øf1H,Øf2H,Øf3H,Øf4H,Øf5H,Øf6H,Øf7H
        db      Øf8H,Øf9H,ØfaH,ØfbH,ØfcH,ØfdH,ØfeH,ØffH

;==============================================================
CLex    db        Ø,   1,   2,   3,   4,   5,   6,   7  ;   Ø
        db        8,   9,  1Ø,  11,  12,  13,  14,  15  ;   8
        db       16,  17,  18,  19,  2Ø,  21,  22,  23  ;  16
        db       24,  25,  26,  27,  28,  29,  3Ø,  31  ;  24
        db       32,  33,  34,  35,  36,  37,  38,  39  ;  32
        db       4Ø,  41,  42,  43,  44,  45,  46,  47  ;  4Ø
        db       48,  49,  5Ø,  51,  52,  53,  54,  55  ;  48
        db       56,  57,  58,  59,  6Ø,  61,  62,  63  ;  56
        db       64,  65,  66,  67,  68,  69,  7Ø,  71  ;  64
        db       72,  73,  74,  75,  76,  77,  78,  79  ;  72
        db       8Ø,  81,  82,  83,  84,  85,  86,  87  ;  8Ø
        db       88,  89,  9Ø,  91,  92,  93,  94,  95  ;  88
        db       96,  65,  66,  67,  68,  69,  7Ø,  71  ;  96
        db       72,  73,  74,  75,  76,  77,  78,  79  ; 1Ø4
        db       8Ø,  81,  82,  83,  84,  85,  86,  87  ; 112
        db       88,  89,  9Ø, 123, 124, 125, 126, 127  ; 12Ø
        db       67,  85,  69,  65,  65,  65,  65,  67  ; 128
```

Listing 1: *(Fortsetzung)*

```
        db       69,  69,  69,  73,  73,  73,  65,  65  ; 136
        db       69,  65,  65,  79,  79,  79,  85,  85  ; 144
        db       89,  79,  85,  36,  36,  36,  36,  36  ; 152
        db       65,  73,  79,  85,  78,  78, 166, 167  ; 16Ø
        db       63, 169, 17Ø, 171, 172,  33,  34,  34  ; 168
        db      176, 177, 178, 179, 18Ø, 181, 182, 183  ; 176
        db      184, 185, 186, 187, 188, 189, 19Ø, 191  ; 184
        db      192, 193, 194, 195, 196, 197, 198, 199  ; 192
        db      2ØØ, 2Ø1, 2Ø2, 2Ø3, 2Ø4, 2Ø5, 2Ø6, 2Ø7  ; 2ØØ
        db      2Ø8, 2Ø9, 21Ø, 211, 212, 213, 214, 215  ; 2Ø8
        db      216, 217, 218, 219, 22Ø, 221, 222, 223  ; 216
        db      224,  83, 226, 227, 228, 229, 23Ø, 231  ; 224
        db      232, 233, 234, 235, 236, 237, 238, 239  ; 232
        db      24Ø, 241, 242, 243, 244, 245, 246, 247  ; 24Ø
        db      248, 249, 25Ø, 251, 252, 253, 254, 255  ; 248

        .CODE
;==============================================================
; IsDigit       : prüft, ob Zeichen in AL eine Ziffer ist
;--------------------------------------------------------------
        Public  IsDigit                         ;
IsDigit Proc    Near                            ;
        cmp     al,'Ø'                          ;
        jb      IsDigit_No                      ;
        cmp     al,'9'                          ;
        ja      IsDigit_No                      ;
IsDigit_Yes:                                    ;
        stc                                     ;
        ret                                     ;
IsDigit_No:                                     ;
        clc                                     ;
        ret                                     ;
IsDigit EndP                                    ;

;==============================================================
; IsAlpha       : prüft, ob Zeichen in AL alphanumerisch ist
;--------------------------------------------------------------
        Public  IsAlpha                         ;
IsAlpha Proc    Near                            ;
        push    ax                              ;
        push    bx                              ;
        mov     bl,al                           ;
        sub     bh,bh                           ;
        mov     al,Byte Ptr CType[bx]           ;
        and     al,UP+LO                        ;
        pop     bx                              ;
        pop     ax                              ;
        jz      IsDigit_No                      ;
        jmp     IsDigit_Yes                     ;
IsAlpha EndP                                    ;

;==============================================================
; ToUpper       : wandelt Zeichen in AL in Großbuchstaben
;--------------------------------------------------------------
        Public  ToUpper                         ;
ToUpper Proc    Near                            ;
        push    bx                              ;
        mov     bl,al                           ;
        sub     bh,bh                           ;
        test    Byte Ptr CType[bx],LO           ;
        je      UprRet                          ;
        mov     al,Byte Ptr CConv[bx]           ;
UprRet: pop     bx                              ;
        ret                                     ;
ToUpper EndP                                    ;

;==============================================================
; ToLex         : wandelt Zeichen in AL in lexikalischen Wert
;--------------------------------------------------------------
        Public  ToLex                           ;
ToLex   Proc    Near                            ;
        push    bx                              ;
        mov     bl,al                           ;
        sub     bh,bh                           ;
        mov     al,Byte Ptr CLex[bx]            ;
        pop     bx                              ;
        ret                                     ;
ToLex   EndP                                    ;
```

Listing 1: *(Ende)*

```
        name    UPPER
        page    65,132
        title   StdIn auf StdOut in Großbuchstaben ausgeben

        DOSSEG
        .MODEL  small
        INCLUDE dos.inc         ; Makros DOS-Calls MASM 5.1
        .STACK  100h

;=================================================================
STDIN   equ     0               ; Handle STDIN
STDOUT  equ     1               ; Handle STDOUT
STDERR  equ     2               ; Handle STDERR

        .DATA
;=================================================================
Buffer          Label   DWord
BufOff          dw      0       ; PufferOffset
BufSeg          dw      ?       ; Puffersegment
BufSize         dw      ?       ; Puffergröße
ReadSize        dw      ?       ; Anzahl gelesene Bytes
QFlag           db      0       ; Flag Anführung

StartMsg        Label   Byte
db      "                                           ",13,10
db      " UPPER 1.00    (C)  1989  G. Jürgensmeier  ",13,10
db      "   Gibt Eingabe in Großbuchstaben aus      ",13,10
db      " Bedienung:                                ",13,10
db      "   UPPER    <Eingabedatei    >Ausgabedatei ",13,10
db      "   Die Zeichen < und > vor den Dateinamen  ",13,10
db      "   müssen eingegeben werden, da sonst auf  ",13,10
db      "   Tastatureingaben gewartet wird.   Dies  ",13,10
db      "   kann mit ^Z oder F6 abgebrochen werden. ",13,10
db      "                                           ",13,10
db      13,10
StartLen        equ     $-StartMsg

DosMsg  db      "UPPER: DOS-Version zu alt.",13,10,7
DosLen  equ     $-DosMsg
ReadMsg db      "UPPER: Fehler beim Lesen.",13,10,7
ReadLen equ     $-ReadMsg
WriteMsg db     "UPPER: Fehler beim Schreiben.",13,10,7
WriteLen equ    $-WriteMsg

        .CODE
;=================================================================
Upper:                          ;
        mov     ax,@data        ; Daten-Segmentreg. init.
        mov     ds,ax           ;
        cli                     ; Interrupte aus
        mov     ss,ax           ; SS und
        mov     sp,Offset STACK ; SP relativ zu DGROUP
        sti                     ;
                                ; Speicherverwaltung einrichten
        mov     bx,sp           ; Stapelzeiger in Paragraphen
        mov     cl,4            ; um Stapelgröße zu erhalten
        shr     bx,cl           ;
        add     ax,bx           ; SS addieren = Programmende
        mov     bx,es           ; Beginn des Programms
        sub     ax,bx           ; Start von Ende abziehen
        @ModBlok ax             ; Speicher über Prg freigeben
                                ; Speicher für Puffer zuweisen
        @GetBlok 0FFFh          ; Versuchen, 64 K zu bekommen
        mov     BufSeg,ax       ; Puffer-Segment speichern
        mov     cx,4            ;
        shl     bx,cl           ; in echte Länge umwandeln
        mov     BufSize,bx      ; wirkliche Länge speichern
                                ; DOS-Version überprüfen
        @GetVer                 ; DOS feststellen
        cmp     al,2            ; DOS 2.0 wird benötigt
        jge     Banner          ;
                                ; DOS-Version zu alt
        @Write  DosMsg,DosLen,STDERR    ; Fehler auf STDERR
        @Exit   1               ; Beenden mit Fehler 1

;-----------------------------------------------------------------
Banner:                         ; Startmeldung
        @Write  StartMsg,StartLen,STDERR
```

Listing 2: *Das Beispielprogramm ToUpper*

```
Schleife:                               ;
        push    ds                      ;
        @Read   Buffer,BufSize,STDIN    ; von STDIN lesen
        pop     ds                      ;
        jc      ReadErr                 ; Lesefehler
        or      ax,ax                   ; Daten gelesen ?
        jz      Exit                    ; Nein -->
        mov     ReadSize,ax             ; AX = Anzahl Bytes
        mov     cx,ax                   ; nach CX
                                        ;
        call    Cvt                     ; konvertieren
                                        ;
        push    ds                      ;
        @Write  Buffer,ReadSize,STDOUT;
        pop     ds                      ;
        jc      WriteErr                ; Schreibfehler
        jmp     Schleife                ; weiterlesen bis Dateiende
Exit:                                   ;
        @Exit   0                       ; Beenden, kein Fehler

;-----------------------------------------------------------------
ReadErr:                                ; Fehler beim Lesen
        @Write  ReadMsg,ReadLen,STDERR  ; Fehler auf STDERR
        @Exit   2                       ; Beenden mit Fehler 2

;-----------------------------------------------------------------
WriteErr:                               ; Fehler beim Schreiben
        @Write  WriteMsg,WriteLen,STDERR ; Fehler auf STDERR
        @Exit   3                       ; Beenden mit Fehler 3

;=================================================================
; Cvt:
;
;       Wandelt im Segment "BufSeg" in der Länge CX
;       alle Klein- in Großbuchstaben, wenn sie
;       nicht in Anführungszeichen stehen.
;-----------------------------------------------------------------
Cvt     Proc    Near                    ;
        push    ax                      ; verwendete Register retten
        push    cx                      ;
        push    si                      ;
        push    es                      ;
        mov     ax,BufSeg               ;
        mov     es,ax                   ;
        mov     si,0                    ;
CvtLoop:                                ;
        mov     al,es:[si]              ; ein Zeichen lesen
        cmp     al,'"'                  ; " prüfen
        jne     CvtNQuote               ;
        not     QFlag                   ;
        jmp     Short CvtLoopEnd;
CvtNQuote:                              ;
        cmp     QFlag,0                 ; nur außerhalb Anf.
konvertieren
        jne     CvtLoopEnd              ;

        call    ToUpper                 ;
        mov     Byte Ptr es:[si],al

CvtLoopEnd:                             ;
        inc     si                      ;
        loop    CvtLoop                 ;

        pop     es                      ;
        pop     si                      ;
        pop     cx                      ;
        pop     ax                      ;
        ret                             ;
Cvt     EndP                            ;

;=================================================================
        INCLUDE is.asm

;=================================================================
        end     Upper
```

Listing 2: *(Ende)*

Bilder einer Ausstellung:

Hard- und Softwareperspektiven 1989

Erstmals veranstaltete Microsoft anstelle der bislang üblichen Herbst-Roadshow für seine Partner ein zentrales Symposium. Fachhändler, Distributoren, Großkunden, Softwareentwickler und Kollegen aus dem Hardwarebereich hatten Gelegenheit, am 28. November 1988 in Frankfurt am Main anerkannte Fachleute zu Strategien und technologischen Perspektiven unserer Branche für die nächsten Jahre zu hören und zu befragen. Unbestrittener Höhepunkt jedoch war die Ausstellung, auf der knapp zwei Dutzend Aussteller marktreife Anwendungen für Microsoft Windows und Microsoft Excel vorführten.

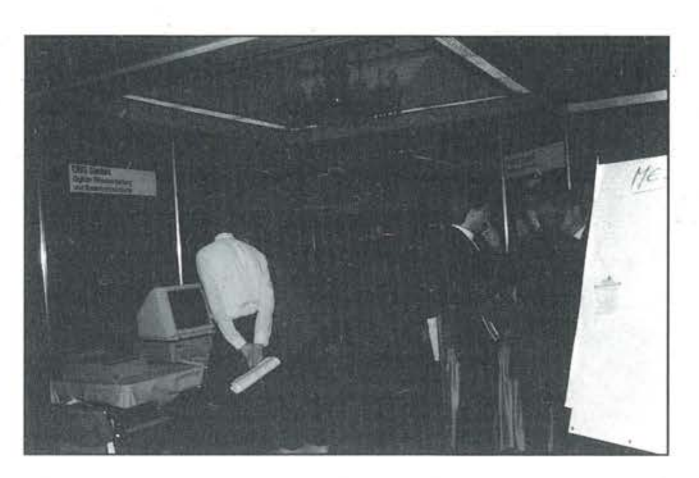


BILD 1: *Rund zwei Dutzend Aussteller zeigten in Frankfurt ihre Anwendungen auf der Basis von Microsoft Windows oder Excel.*

Excel: Software des Jahres

Das Jahr 1988 war ein Windows-Jahr. Nicht nur, daß sich die im Vorjahr eingeführte Version 2.0 dieser MS-DOS-Betriebssystemerweiterung auf dem Markt schnell durchsetzen konnte, mit Microsoft Excel betrat auch die erste große Standardanwendung aus dem Haus Microsoft die Windows-Bühne. Dieses Tabellenmanagementprogramm war die erste Anwendung, die die Leistungsfähigkeit von Windows richtig ausspielen konnte. Als Tabellenmanagementprogramm mit integrierten Grafik- und Datenbankfunktionen gelang es Excel binnen weniger Monate, zum Inbegriff einer neuen Softwaregeneration zu werden. Kein anderes Tabellenkalkulationsprogramm war wie Excel in der Lage, Kritiker zu begeistern. Dynamischer Datenaustausch mit externen Anwendungen, professionelle Layout- und Darstellungsmöglichkeiten, bislang unbekannte Funktionstiefe, ausgereifte Hilfe- und Computer Based Training-Module und eine Makrosprache, die Amateuren eine schnelle Anpassung an individuelle Anforderungen und Profis die Erstellung komplexer Branchenlösungen ermöglicht - das waren die vielbesprochenen Highlights. So errang Excel zahlreiche Preise und Auszeichnungen sowohl in den USA als auch in Deutschland. Erst vor wenigen Wochen wählten die Redaktionen zehn europäischer PC-Fachzeitschriften Microsoft Excel zur Software des Jahres. In Deutschland machte Excel Microsoft schon zwei Monate nach Auslieferung zum größten Spreadsheetanbieter.

Der Erfolg von Excel verdankt sich aber nicht nur der technologischen Überlegenheit des Produkts. Alle haben an diesem Erfolg mitgeholfen: Handel, Schulungshäuser und Entwickler. Zahlreiche Marketingaktionen wurden zur Markteinführung durchgeführt. Der Distributor Computer 2000 bundelte sein Above-Board mit Excel. Die Zeitung PC Magazin organisierte für Entwickler einen Programmierwettbewerb für Excel-Anwendungen. In zahlreichen Fachgeschäften warben die Excel-Plakate von Microsoft. Das Harlekin-Motiv errang als Zeitungsanzeige Preise für kreatives Marketing.

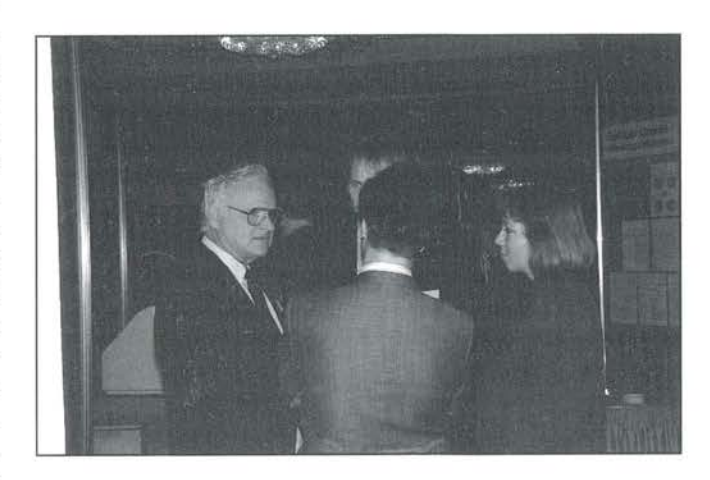


Bild 2: *Fachsimpeln am Rande der Ausstellung: Emil Widmer (links), Inhaber der Schweizer »Computertechnik für Manager« im Gespräch mit Mitarbeitern der Microsoft GmbH, unter ihnen Karin Paul (rechts), Leiterin Sales Distributors bei Microsoft.*

Windows hat sich durchgesetzt

Excel ist aber nicht nur ein erfolgreiches Produkt. Es wirbt auch mit seiner Existenz für Microsoft Windows. Windows ist - endlich! - ins Gerede gekommen, und zwar im positiven Sinn. Die Version 2.0 hat viele Mängel der alten Version 1.x ausgeräumt. Windows ist schneller und flexibler geworden. Windows 386 schließlich konnte sich zum Standard unter den Betriebssytemerweiterungen für 386er Rechner entwickeln. Seine Multitaskingfähigkeiten bewegten viele Hardwarehersteller dazu, Windows 386 gleich mit ihren Rechnern zusammen auszuliefern. Compaq, IBM, Zenith, Victor, um nur einige zu nennen, führten erfolgreich solche Aktionen durch. Unterstützt wurde der Erfolg von Windows sicherlich durch die Diskussion um MS OS/2, das neue

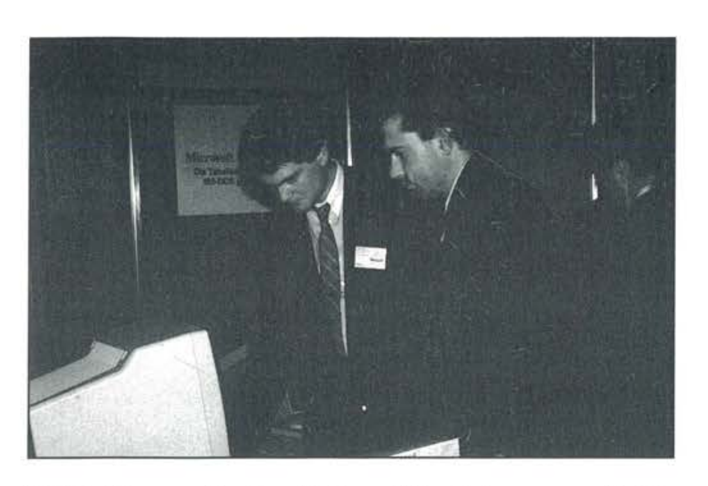


Bild 3: *Gregory Gordon, Microsoft-Produktmanager für Macintosh-Software, erläuterte an seinem Informationsstand dem Publikum Ähnlichkeiten und Unterschiede zwischen den Excel-Versionen für MS-DOS unter Windows und den Macintosh-Rechnern.*



Bild 5: *Frank Utermöhlen, Microsoft Produktmanager für Compiler, zeigte neueste MS-OS/2 Compilertechnologie.*



Bild 4: *Meist dicht umlagert: Der Stand von Siemens Österreich, auf dem Computer Aided System Engineering mit dem Windows-Produkt EasyCASE demonstriert wurde.*



Bild 6: *Gerhard Rutkowski, Leiter von der Abt. Officeland der Münchner Softwareschmiede Softlab bei einer kleinen Energiespende für den Düsseldorfer Microsoft-Mitarbeiter Lothar Splittstösser.*

Betriebssystem von Microsoft und IBM. MS OS/2 bzw. BS/2, wie es bei IBM heißt, wird in der Version 1.1 zusammen mit dem Presentation Manager ausgeliefert werden. Dieser Presentation Manager wird das gleiche Gesicht haben wie Microsoft Windows. Die Bedieneroberfläche der neuen Version 4.0 von MS-DOS kommt als drittes System hinzu. Damit haben es Anwender künftig bei zeichenorientierten Programmen unter MS-DOS, bei grafikorientierten MS-DOS Windows-Applikationen und bei MS OS/2-Applikationen unter dem Presentation Manager mit der gleichen Oberfläche zu tun. Mit dem Presentation Manager/X, im November 1988 angekündigt, können sich sogar UNIX-Anwender diesem Anwenderkreis anschließen. Typischer »Macintosh-Komfort« fand durch Windows Eingang in die Welt des Industriestandards.

Diese Perspektive hat sicherlich viele Anwender im vergangenen Jahr bewogen, auf Microsoft Windows umzusteigen. Heute werden jeden Monat weltweit mehr Windows-Packungen verkauft als Macintosh-Rechner.

Und auch Systementwickler wandten sich verstärkt Windows zu. Die gemeinsamen Programmierschnittstellen von Windows, Presentation Manager und Presentation Manager/X erlauben ihnen die Erstellung von Anwendungen, die sich ohne größere Umbaumaßnahmen für MS-DOS, MS OS/2 und UNIX anpassen lassen. Und selbst der Schritt zu Macintosh-Anwendungen ist für den Windows-Entwickler ein Katzensprung. Eben dies bewiesen die Aussteller auf dem Frankfurter Microsoft Symposium. Kaum einer von ihnen entwickelt ausschließlich für Windows. Fast jeder hat zugleich MS OS/2 oder den Macintosh im Visier.



Bild 7: *Dr. Michael Müller, Leiter des Großkundenvertriebs bei Microsoft, im Gespräch mit Barbara Fischer, die Microsofts Kunden in der Schweiz betreut.*



Bild 9: *Einen Blick auf die brandneue nächste MS OS/2-fähige Version 5.0 von Microsoft Word konnten Besucher am Stand von Karin Süsser, Produktmanagerin für Textverarbeitung bei Microsoft, riskieren. Bis zur Marktreife und Auslieferung freilich bleiben noch einige Wochen Zeit.*



Bild 8: *Technologische Avantgardeleistungen waren am CD ROM-Stand von Bertelsmann zu begutachten.*



Bild 10: *Ein CAD-System mit integrierter Datenbank auf Windows-Basis wurde auf dem Stand von CAS gezeigt.*

Die Aussteller

Zu den renommiertesten Ausstellern zählte sicherlich die österreichische Siemens AG. In Wien hat man schon lange Erfahrungen mit Microsoft Windows sammeln können. So wurde mit EasyCASE ein Produkt vorgestellt, daß seine Markteinführung gerade vor sich hat, seine Marktreife aber schon auf der Ausstellung hinlänglich beweisen konnte. Dabei handelt es sich bei EasyCASE eigentlich um drei Programme. EasyCASE (SD) unterstützt das System Designing mit Kommunikationsplänen, EasyCASE (DD) bildet ein Data Dictionary für Standardauswertungen, und EasyCASE (SP) erleichtert als Editor strukturiertes Programmieren mit Struktogrammen.

Entwickler mit MS OS/2-Ambitionen versammelten sich um den Microsoft Compiler-Stand. Dort konnte man sich BASIC-, FORTRAN-, C-, Assembler- und Pascal-Compiler für MS OS/2 ansehen. Die größte Aufmerksamkeit freilich erregte der neue Microsoft COBOL 3.0-Compiler. Ihn hatte Microsoft erst wenige Tage zuvor erstmals in Deutschland vorgestellt. Dabei handelt es sich um eine Gemeinschaftsentwicklung von Microsoft und Micro Focus.

Zu den renommierten Entwicklern, die zugleich für Windows und den Presentation Manager programmieren, gehört die Firma Softlab. Dieses Unternehmen, das in Deutschland zu den bedeutendsten Entwicklern von Branchenlösungen gerechnet wird, nutzt vor allem die SAA-Schnittstellen von Windows, mit denen ein Übergang zu den Mini- und Großrechnern der IBM-Welt geschaffen wird.



Bild 11: *Informationsgespräch auf dem Stand der Management Software AG.*



Bild 13: *Christian Wedell, Geschäftsführer der Microsoft GmbH, läßt sich von Peter Tewes von der Firma MICRO SERVICE eine Excel-Anwendung erklären.*

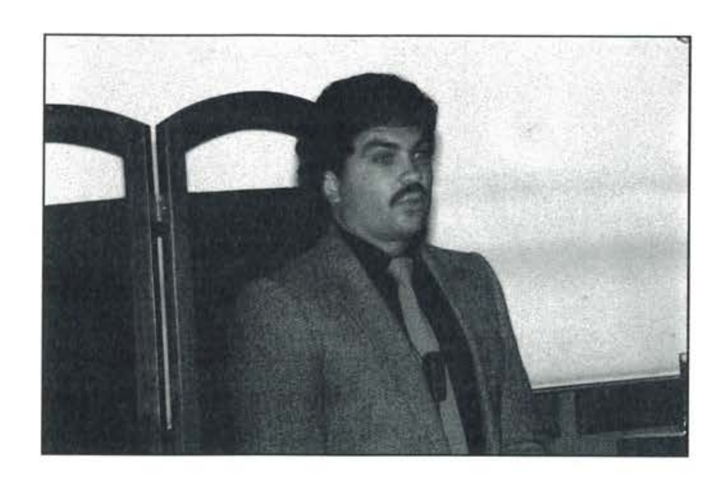


Bild 12: *Kevin P. Welch, Präsident der Eikon Systems, übernahm auf seinem Stand selbst die Regie.*

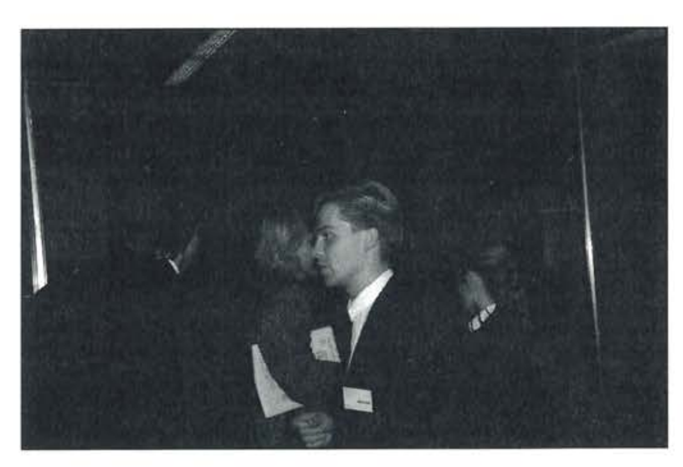


Bild 14: *Ralf Klocke, Microsoft Verkaufsleiter für den Händlervertrieb, war ein gefragter Mann auf der Ausstellung. Schließlich waren »seine« Händler zahlreich erschienen.*

Auf lange und erfolgreiche Arbeit mit Windows-Produkten kann die Firma GCA verweisen, die ebenfalls mit einem eigenen Stand auf der Frankfurter Messe vertreten war. Derzeit unternimmt sie eine Vielzahl von Marketinganstrengungen, um das Produkt »Accessories No.1« im Markt zu etablieren - und wie es scheint, mit großem Erfolg. Die Accessories sind eine Sammlung von Zusatzprodukten, die Windows-Anwendern das Leben leichter machen können. So lassen sich eigene Menüs mit grafischen Symbolen für unterschiedlichste Dateien anlegen. Mit dem Bridge-Modul kann man GEM-Formate in Windows umsetzen. Ein anderes Modul erlaubt die Anfertigung von Screencopies. Weitere Programme erhöhen die Datensicherheit, erleichtern die Druckerkontrolle, bieten eine digitale Bildschirmuhr und ermutigen den Spieltrieb von Windows-Anwendern. Neu war das Programm FORM MASTER auf dem GCA-Stand. Mit FORM MASTER lassen sich Formulare anfertigen und komfortabel bearbeiten. Dabei werden eine Vielzahl von Formatierungsmöglichkeiten geboten. Von diesem Produkt werden in den nächsten Wochen sicherlich noch zahlreiche Testberichte in der einschlägigen Fachpresse zu lesen sein.

Für viele Besucher überraschend war der Auftritt von Bertelsmann auf der Ausstellung. Bertelsmann gilt in der Branche als Vorreiter der CD ROM-Technologie. Bisher wurden unter anderem folgende Anwendungen dieser neuen optischen Speichertechnologie vorgestellt: Washington Presstext (Informationen der US-amerikanischen Regierung), Liefern & Leisten (Das Deutsche Branchen-Fernsprechbuch), Silverdat (Datensystem für die Auto-



Bild 15: *Vincenz Wirth (links hinten), Leiter der Produktmanagment-Gruppe für Systemsoftware und Programmiersprachen bei Microsoft, mußte zahlreichen Besuchern Rede und Antwort zum gerade fertiggestellten MS OS/2 Presentation Manager stehen.*

mobilwirtschaft), Lauer-Taxe (Datenbank für den pharmazeutischen Bereich), Wer liefert was? (Bezugsquellennachweis für den Einkauf), Die Bibel, Lawbase (Entscheidungen des Schweizer Bundesgerichts), Twixtel (Schweizer Telephonbuch), Datenbank Leitverzeichnisse (Orts- und Straßenverzeichnis der Deutschen Bundespost und Müllers großes Deutsches Ortsbuch). Was aber hat all das mit Windows zu tun? Ganz einfach: Seit 1988 unterstützt das Bertelsmann Retrieval (Zugriffs)-System COBRA Microsoft Windows. Im Prospekt des BCB - Bertelsmann Computer Beratungsdienstes heißt es denn mit Blick auf MS OS/2 auch:

»Seit 1988 können Ihre Anwendungen auch von Microsoft Windows unterstützt werden. So realisieren wir bereits jetzt für Sie die Anbindung an künftige Standard-Betriebssysteme und den Presentation Manager.«

Das IBM Personal Publishing System wurde auf dem Stand der KD COMPUTER FORUM GmbH vorgeführt. Bei diesem DTP-Komplettpaket von Mother Blue handelt es sich ebenfalls um ein auf Microsoft Windows basierendes System. Die Hardware - Personalcomputer, Laserdrucker - kommt von IBM, das DTP-Programm von Aldus (Pagemaker), die »Seele« von Microsoft: Windows.

Die Pirmasenser Computer Anwendungs- und Systemberatung CAS zeigte ihr neues Paket COSTING. COSTING nutzt die Windows-Oberfläche für ein integriertes Grafik-/Kalkulationsmodell. Es dient der Kostenkalkulation in den CAD-Abteilungen der Schuh- und Bekleidungsindustrie.

Je eine Windows-Applikation zur Auftragsbearbeitung bei Zimmereibetrieben und für Maler wurde von der EDV-Beratung THEOBALD vorgestellt. Dieses Unternehmen hat bislang rund 400 verschiedene Branchenlösungen entwickelt und vermarktet. Seit einiger Zeit befindet man sich nun im Umstieg auf die Windows-Entwicklung.

Warum man bei THEOBALD trotz programmiertechnischer Startschwierigkeiten auf Windows umgestiegen ist, konnte man am Stand erfahren: »Von den bis jetzt 30 Anwendern möchte niemand das Programm missen. Anwender aller Altersschichten, bei älteren Anwendern hatten wir ursprünglich erhebliche Bedenken, kommen mit dem Programm prima klar. Programme mit konventioneller Bedieneroberfläche werden von diesen Kunden nicht mehr akzeptiert. Die Einarbeitung in das Programm ging bis auf drei Fälle schneller, als wir es von konventionellen Programmen gleichen Leistungsumfangs her gewohnt waren. Als riesigen Vorteil sehen wir die quasi genormte Bedieneroberfläche. Wenn man ein MS-WINDOWS Programm bedienen kann, kann man alle MS WINDOWS Programme bedienen.« Der Umstieg auf den Presentation Manager ist übrigens auch bei THEOBALD schon geplant.

Eine komplette Windows-Programmbibliothek für das technische Laborwesen im Straßenbau wurde von der Firma bpi-Büro für Planung und Ingenieurtechnik gezeigt. Die verschiedenen Module dienen unter der einheitlichen Windows-Oberfläche zum Beispiel der Verwaltung des technischen Regelwerks und möglicher Rezeptkomponenten für bituminöse Mischgüter, der Eignungsprüfung solcher Mischungen, der Durchführung und statistischen Auswertung von Mischgutuntersuchungen und der Überwachung von Mineralstoffen und Bindemitteln - woran man unschwer erkennen kann, daß Microsoft Windows zum ungeahnten Wegbereiter für viele geworden ist!

Was gab es noch zu sehen?

- Die Firma Eikon Systems aus den USA gehört zu den ältesten Windowsanbietern. Sie konnte ein komplettes Sortiment von Produkten vorführen.
- Das Terminplanungssystem »TIMEGRAF« von netnice & partner bewies die Eignung grafischer Oberflächensysteme für Netzplanapplikationen.
- »artus« von der Berliner Firma TRONY, entwickelt für die Nachbearbeitung gescannter Bilder.
- Die Gesellschaft für angewandte Informatik aus Karlsruhe zeigte Windows-Software zur Meßdatenerfassung.
- Integrata und Olivetti Bildungszentrum stellten ihre Schulungsaktivitäten für Windowsentwickler und -anwender vor.

Natürlich gab es auch eine Reihe von Excel-Anwendungen zu begutachten. Immer mehr Windows-Entwickler greifen auf dieses Tabellenmanagementprogramm mit seiner entwickelten Makrosprache als Entwicklungstool zurück. Die Firma MICRO SERVICE zum Beispiel zeigte gleich mehrere Excel-Anwendungen, unter anderem zur Lagerverwaltung und zum Rechnungswesen. Bei MICRO SERVICE fährt man zweigleisig. Die in der Entwicklung erworbenen Excel- und Windows-Kenntnisse werden gleich in Excel-Schulungen und -Seminaren weitergegeben.

Microsoft COBOL 3.0 ist da:

Die Brücke zwischen DOS, OS/2 und Mainframes

Microsoft bietet ein neues leistungsfähiges Werkzeug zur Entwicklung und Wartung von Mainframe-Applikationen in einer PC-Umgebung an: die Version 3.0 des optimierenden COBOL-Compilers. Der durch das »US National Bureau of Standards« freigegebene Compiler entspricht dem ANSI 85-COBOL High Level-Standard. Er unterstützt den großen Speicherbedarf bei Mainframe-Applikationen, erzeugt Code, der zehnmal schneller ist als derjenige von COBOL 2.2 und ist kompatibel mit den meisten Mainframe-, Mini- und Mikrocomputer-COBOL-Dialekten, einschließlich der IBM-Mainframe-COBOL-Compiler.

Microsoft COBOL erlaubt zusammen mit MS-OS/2 die Übernahme von Großrechner-Applikationsprogrammen für den Einsatz in PC-Umgebungen. Durch die Kompatibilität von Microsoft COBOL 3.0 zu den meistverbreiteten Mainframe-, Mini- und Mikrocomputer-COBOL-Dialekten, zum Beispiel zu IBM VS COBOL-II, IBM OS/VS-COBOL, X/Open COBOL, Data-General-COBOL, RM/COBOL und Microsoft COBOL 2.2, kann der Entwickler ohne Probleme von einer Umgebung in die andere wechseln. Diese Möglichkeit wird darüber hinaus noch dadurch untermauert, daß Microsoft COBOL 3.0 dem SAA (System-Applikations-Architektur)-Standard der IBM entspricht.

Unter MS-OS/2 werden Applikationen unterstützt, die bis zu 16 Mbyte virtuellen Speicher adressieren. Diese Fähigkeit setzt die Software-Entwickler in die Lage, größte Mainframe-Applikationsprogramme auf Personalcomputern laufen zu lassen, ohne daß sie durch die bisherige Speicherbegrenzung unter MS-DOS eingeschränkt sind.

Die leistungsfähigen Entwicklungswerkzeuge des Microsoft COBOL 3.0 bilden einen kompletten Satz sowohl für MS-DOS als auch für MS-OS/2-Applikationen.

Eingeschlossen ist dabei der ANIMATOR - ein Source-Level-Debugger, der ganz auf den COBOL-Programmierer zugeschnitten ist und Source-Level-Tracing, Backtracking, Breakpoints, DO-Statements, periodische Breakpoints und die Fähigkeit beinhaltet, einzelne COBOL-Anweisungen im Online-Modus direkt auszuführen (*Bilder 1 und 2*). Ebenfalls Bestandteil der Entwicklungswerkzeuge ist der Microsoft-Editor, eine MS-DOS Real-Mode und MS-OS/2 Protected-Mode-Editierumgebung mit vollständiger Rekonfigurierbarkeit sowie Makro-Unterstützung.

COBOL 3.0 ist ein »Maschinencode«-Compiler, der sehr schnellen Code erzeugt. Applikationen, die mit Microsoft COBOL 3.0 compiliert wurden, laufen zehnmal schneller als Applikationen, die mit der Version 2.2 compiliert wurden. Die I/O-Geschwindigkeit ist um 30 Prozent höher.

Gemäß dem ANSI 85 High-Level-Standard beinhaltet Microsoft COBOL 3.0 Funktionen zur strukturierten Programmierung, wie In-Line-PERFORM, EVALUATE, Scope Terminator, negierte Bedingungen, INITIALIZE, Global-Referenzen und Referenz-Modifikationen; Funktionen, die zur Verbesserung der Software-Wartung beitragen. Als Teil der Sprachenfamilie für MS-DOS und MS-OS/2-Systeme unterstützt COBOL 3.0 das Aufrufen von Subroutinen, die in anderen Microsoft-Sprachen geschrieben wurden. Der Compiler bietet außerdem die Möglichkeit, daß Applikationen, die unter MS-DOS geschrieben wurden, nun auf einfache Weise in gängigen LANs eingesetzt und weiterentwickelt werden können. Durch die Einsatzfähigkeit von COBOL 3.0 unter MS-OS/2 lassen sich Anwendungen extrem einfach in diese neue Umgebung übernehmen und die Vorteile der virtuellen Speicherverwaltung sowie des Multitaskings nutzen.

Strategische Zusammenarbeit mit Micro Focus

Im Zusammenhang mit der Ankündigung des COBOL 3.0-Compilers gab Microsoft die strategische Marketing- und Software-Entwicklungs-Allianz mit Micro Focus bekannt, deren Ergebnis das Produkt ist.

```
 80 play-game section.
 81 play-1.
 82     perform with test after
 83         until char not = "Y" and char not = "y"
 84         call clear-screen
 85         display
 86             "To select a square type a number between 1 and 9"
 87             upon crt
 88         perform init
 89         move "Shall I start ? " to question
 90         perform get-reply
 91         if char = "Y" or char = "y"
 92             move 10 to check(5)
 93             perform put-move
 94         end-if
 95         perform new-move until game not = spaces
 96         move "Play again ?    " to question
 97         perform get-reply
 98     end-perform.
 99
100 play-stop.
Animate-TICTAC--------------------------Level=01-Speed=5-Ins-Caps-Num-Scroll
F1=help F2=view F3=align F4=exchange F5=where F6=look-up  F9/F10=word-</> Escape
Step Go Zoom next-If Perform Reset Break Env Query Find Locate Text Do 0-9=speed
```

Bild 1: *Debuggen mit dem Animator von MS-COBOL 3.0.*

```
Help screen for...              Animate           Page 2            Help82

F1=help           Display previous screen |next-If       Execute until next IF
F2=view           Display user screen     |Perform-level Set executed perform level
F3=align          Set this line to 3      |Reset         Reset execution position
F4=exchange       Move to other screen    |Break         Set/unset break-points
F5=where          Find current position   |Env           Set execution environment
F6=look-up        Set entered line to 3   |Query         Examines data-item
F9=word-left      Move one word to left   |Find          Find next occurrence
F10=word-right    Move one word to right  |Locate        Locate declaration of item
Escape            Leave Animator          |Text          Set screen separator
Step              Execute one instruction |Do            Execute typed COBOL syntax
Go                Execute slowly          |0-9           Set default Go speed
Zoom              Execute at full speed   |

                       press F1 or space bar to return
Animate-TICTAC--------------------------Level=03-Speed=5-Ins-Caps-Num-Scroll
F1=help F2=view F3=align F4=exchange F5=where F6=look-up  F9/F10=word-</> Escape
Step Go Zoom next-If Perform Reset Break Env Query Find Locate Text Do 0-9=speed
                                                                    ]
```

Bild 2: *Beim Debuggen stehen viele Befehle zur Verfügung.*

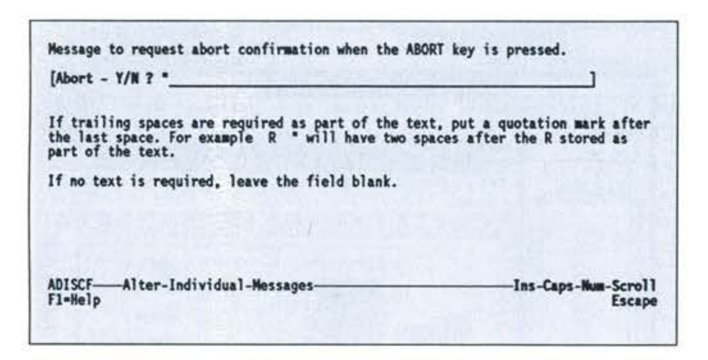


Bild 3: *Mit einem Konfigurationsprogramm kann das Bildschirmmodul von MS-COBOL 3.0 angepaßt werden.*

Micro Focus ist eines der weltweit führenden Unternehmen in der COBOL-Technologie mit Firmensitzen in Newbury (England), Palo Alto (Kalifornien) und München. Die Zusammenarbeit mit Micro Focus wird die Position von COBOL in der Mikrocomputer-Entwicklungsszene stärken, weil Microsoft sein führendes Know-How bei Entwicklungswerkzeugen und moderner Software-Technologie und Micro Focus seine hervorragenden PC-COBOL-Produkte einbringt, so daß der Anwender von den Stärken beider Unternehmen profitiert.

»Microsoft COBOL 3.0 ist eine deutliche Demonstration des Engagements von Microsoft in der DP/MIS-Welt«, so Christian Wedell, Geschäftsführer der deutschen Microsoft GmbH, zur Ankündigung der neuen COBOL-Version. »Es ist sicher, daß die neue Generation der 80286/80386-Personalcomputer eine lebensfähige Grundlage für die Entwicklung von DP/MIS-Applikationen auf MS-DOS- und MS-OS/2-Systemen darstellt. Wir engagieren uns dabei mit einem Hochleistungs-COBOL-Compiler, der ein integraler Bestandteil unserer Familie von MS-DOS- und MS-OS/2-Sprachen und -Werkzeugen ist«, so Wedell. Und: »Die strategische Verbindung zwischen Microsoft und Micro Focus ist eine Zusammenführung der weltweit führenden COBOL-Technologie mit der führenden Mikrocomputer-System-Technologie.« Die beiden Unternehmen bieten nun nach Meinung von Wedell eine Entwicklungslösung für das größte Segment von Entwicklern innerhalb der Software-Branche: die COBOL-Applikationsentwickler.

Daten und Fakten

Microsoft COBOL 3.0 wird bereits ausgeliefert. Für Besitzer der Version 2.2 hält Microsoft ein Upgrade-Angebot bereit. Der neue Compiler läuft auf IBM-PCs oder 100%-Kompatiblen unter MS-OS/2 und MS-DOS (3.0 und höher). Systemvoraussetzungen sind ein doppelseitiges Disketten-Laufwerk, ein Festplatten-Laufwerk und ein Hauptspeicher von 1,5 Mbyte (MS-OS/2) oder 384 Kbyte (MS-DOS).

pi

Windows besser nutzen mit Expanded Memory:

Leistungssteigerung durch EMS

1981 führte IBM seinen Personalcomputer ein, der den Entwicklern von Anwendungssoftware unter anderem mehr Speicher zur Verfügung stellte. Sie werden sich sicher erinnern, daß die damals am meisten verkaufte Maschine der Apple II war, der mit seiner 6502-CPU 64 Kbyte Speicher adressieren konnte. Der Personalcomputer von IBM war mit seinem Intel 8088-Microprozessor in der Lage, 1 Mbyte zu adressieren. Wenn man den für die Hardware reservierten Speicherbereich abzieht, war der Speicher von 640 Kbyte, der der Anwendung zur Verfügung stand zehnmal so groß, wie beim Apple. Plötzlich hatten Programmierer und Anwender mehr Speicher, als sie benötigten.

Die Benutzer stießen an die Grenze von 640 Kbyte. Die Software wurde durch neue Möglichkeiten größer, Tabellenkalkulation und Textverarbeitung wuchsen durch die Anforderungen der Anwender, und durch die Fortschritte bei der Halbleiterfertigung sank der Speicherpreis. Was einst als eine ungeheure Größe erschien, war nicht mehr genug.

Um den Anforderungen nach mehr Speicher gerecht zu werden, wurde 1984 die *Expanded Memory Specification* (EMS) eingeführt, um Anwendungen mehr als die 640 Kbyte an Speicher zur Verfügung zu stellen. EMS definiert die Software-Schnittstelle zum Ansprechen von Expanded Memory. Unter Verwendung von EMS kann eine Anwendung bis zu 8 weitere Mbyte an Daten im RAM speichern. AST Research Inc. entwickelte eine Obermenge, die *Enhanced Expanded Memory Specification* (EEMS) genannt wird. Sie bietet größere Flexibilität in der Richtung, daß das Expanded Memory nur für den Gebrauch der Anwendung eingeblendet wird.

Die EMS- und EEMS-Spezifikationen wurden von Lotus, Intel und Microsoft im August 1987 erweitert zur Expaned Memory Specification 4.0 (LIM EMS 4.0). EMS 4.0 bietet bis zu 32 Mbyte Expanded Memory, also viermal mehr als die unter EMS 3.2 möglichen 8 Mbyte. Durch die Vereinigung zweier ähnlicher, aber doch unterschiedlicher Speicher-Spezifikationen bietet LIM EMS 4.0 eine vereinfachte Speicherunterstützung für Windows, da jede EMS- und EEMS-Karte zu LIM EMS 4.0 kompatibel gemacht werden kann, indem ein Treiber hinzugefügt wird. Neben der Vereinheitlichung der Expanded Memory-Schnittstelle bietet EMS 4.0 auch Multitasking-Unterstützung für Expanded Memory.

Dieser Artikel beschreibt, wie Expanded Memory arbeitet, wie Windows 2.0 Expanded Memory benützt, wie virtueller Speicher im Windows/386 implementiert ist, und wie die Unterstützung von Expanded Memory Ihre Windows-Programmierung beeinflussen kann.

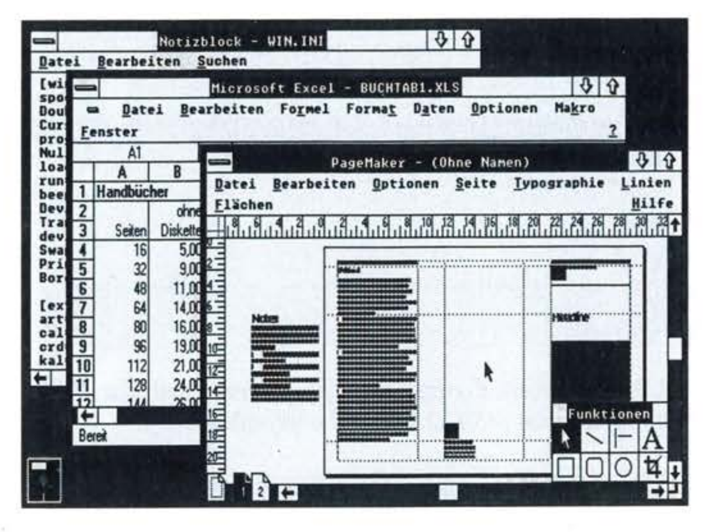


Bild 1: *EMS ermöglicht es dem Benutzer mehrere große Anwendungen, wie Microsoft Excel und Aldus PageMaker, unter Windows laufen zu lassen.*

Expanded Memory

EMS 4.0 erlaubt der Anwendung den Zugriff auf mehr als 640 Kbyte Speicher. Hierzu wird ein Teil des 1 Mbyte-Adreßbereichs des 8088 als Fenster in das Expanded Memory verwendet. Dieses Fenster wird Seitenrahmen genannt. Abhängig von der Nutzung des Systemspeichers, reicht der Seitenrahmen von 16 bis 1024 Kbyte.

Der Seitenrahmen ist in physische Seiten unterteilt, die typischerweise 16 Kbyte groß sind. Das Expanded Memory ist in logische Seiten unterteilt, die ebenso 16 Kbyte groß sind. Die Anwendungen erhalten den Speicher, indem sie den Expanded Memory Manager (EMM) die Anzahl der benötigten logischen Seiten allokieren lassen. Die Anwendung muß den EMM einen Satz logischer Seiten in den physischen Adreßbereich einblenden lassen, bevor sie diese benutzen kann. Die Anwendung kann dann das Expanded Memory wie konventionellen Speicher benutzen. Beim Zugriff auf einen anderen Satz von logischen Seiten, muß der EMM diese vorher einblenden. Eine Anwendung kann auf keine Seite zugreifen, die nicht vorher eingeblendet wurde. Obwohl der physische Adreßbereich 1 Mbyte groß ist, kann das Expanded Memory bis zu 32 Mbyte groß sein.

Bild 2 ist ein Speicherabbild, das die Beziehung einer MS-DOS-Anwendung, wie etwa Lotus 1-2-3, zu den logischen Seiten des Expanded Memory zeigt. In diesem Beispiel existieren zwei Rahmen im physischen Adreßbereich, genannt *Rahmen 1* und *Rahmen 2*. Die zwei logischen Seiten, die *Seite 3* und *Seite 6* genannt werden, sind gerade in die beiden Rahmen eingeblendet. Beachten Sie, daß die doppelt umrandeten Bereiche für das Betriebssystem und die Hardware (z.B. den Bildschirmspeicher) reserviert sind. Der ganze Rahmen befindet sich oberhalb 640 Kbyte, obwohl EMS 4.0 es gestattet, daß das Expanded Memory überall in den physischen Adreßbereich eingeblendet wird.

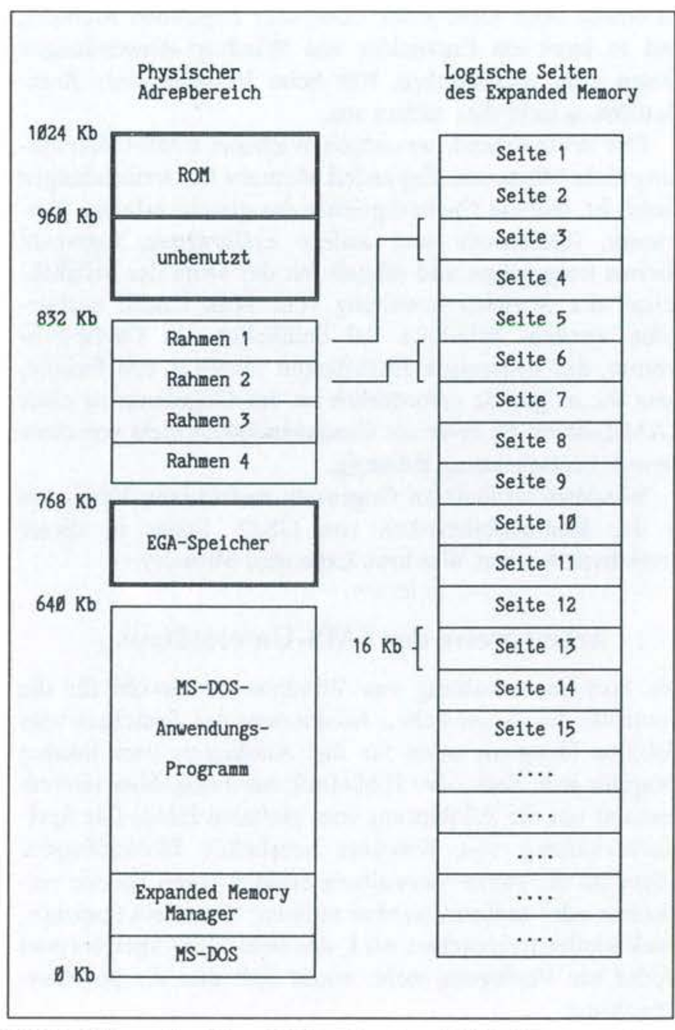


Bild 2: *Dieses Speicherabbild zeigt eine MS-DOS-Anwendung wie Lotus 1-2-3, die Expanded Memory nach EMS 3.2 benutzt. Die Rahmen 1 bis 4 bestehen aus jeweils 16 Kbyte physischem Speicher.*

Expanded Memory benötigt spezielle Hardware und spezielle Software. Die Hardware besteht aus einer Karte, die Speicherbausteine enthält. Die Software ist ein Einheitentreiber, der EMM heißt. Dieser bearbeitet die Anforderungen ans Expanded Memory und läßt die Speicherkarte das Ein- und Ausblenden durchführen.

Die Anwendungen müssen durch einen Aufruf des EMM den Speicher allokieren, und durch einen anderen Aufruf diesen allokierten Speicher in den physischen Speicherbereich einblenden. Die dritte Aktion, die die Anwendung durchführen muß, ist das Freigeben des nicht mehr benötigten Speichers. Diese drei Schritte entsprechen den Aktionen, die beim Allokieren von Speicher von Windows ausgeführt werden. Speicher wird allokiert (entweder mit `GlobalAlloc` oder mit `LocalAlloc`), er wird in den physischen Adreßbereich eingeblendet (mit `GlobalLock` oder mit `LocalLock`), und er wird nach Gebrauch freigegeben (mit `GlobalFree` oder mit `LocalFree`). Der vierte Schritt, den Speicher auszublenden (`GlobalUnlock` oder `LocalUnlock`), ist in EMS Teil des Einblendeaufrufs.

Einige EMS-Karten erlauben nur einen 64 Kbyte großen Block, der über der 640 Kbyte Grenze liegen muß. Andere Karten erlauben viele Seitenrahmen, ohne Beschränkung der Größe und der Lage. Obwohl der EMS-4.0-Einheitentreiber für beide Kartentypen geschrieben werden kann, ist es doch notwendig, daß Windows beide unterschiedlich behandelt. Ältere Karten ließen den Bereich unter 640 Kbyte unberührt und boten einen relativ kleinen Seitenrahmen. Neuere Karten können so eingestellt werden, daß sie alles über 256 Kbyte belegen, was einen Bankbereich von über 600 Kbyte ergibt.

Wie wir später sehen, beeinflußt die Rahmengröße die Art und Weise, wie die Speicherverwaltung von Windows das Expanded Memory benutzt. Ich werde in Zukunft die Version mit einem Rahmen als alte Version, und die mit mehreren Rahmen als neue Version bezeichnen. Auch kann das Vorhandensein von speicherresidenten Programmen (TSR, Terminate and Stay Resident), oder Netzwerkkarten oder anderer Anwendungen mit festem Speicher Windows veranlassen, neuere EMS-Karten im Modus mit kleinem Rahmen zu benutzen.

Die Trennlinie zwischen dem umschaltbaren und dem nicht umschaltbaren Teil der gegebenen EMS-Konfiguration wird auch Umschaltlinie (bank line) bezeichnet. Die Kenntnis, wo Objekte angesiedelt werden, ob unter oder oberhalb der Umschaltlinie, beeinträchtigt Sie nicht, wenn Sie eine ausschließlich allein laufende Windows-Anwendung schreiben. Wenn Sie aber Anwendungen programmieren, die miteinander kommunizieren, Ihre eigene dynamische Linkbibliothek, Windows Hooks benutzen, oder die Zwischenablage bzw. DDE nutzen wollen, so sollten Sie sich mit der Position der Speicherobjekte relativ zur Umschaltlinie näher beschäftigen.

Das Ein- und Ausblenden von logischen Seiten des Expanded Memory erfordert nicht das Kopieren großer Speicherblöcke. Statt dessen werden nur ein paar Register manipuliert. Nachdem diese Register beschrieben sind, ist der Speicher eingeblendet und kann so schnell und effizient angesprochen werden.

Expanded Memory und Windows

Windows Version 2.0 benutzt zum Zwischenspeichern von Anwendungen Expanded Memory. Durch diese Unterstützung des Expanded Memory können große Anwendungen wie Microsoft Excel und Aldus PageMaker gleichzeitig laufen - in einer Geschwindigkeit, als wären sie nur allein vorhanden. Windows 1.x benutzte Expanded Memory, um typische MS-DOS-Anwendungen auszulagern, aber nicht für Windows-Anwendungen. Windows 2.0 unterstützt diese älteren Anwendungen, aber mit dem Vorteil, daß Windows-Anwendungen im Expanded Memory zwischengespeichert werden.

Speicher ist die kostbarste Ressource unter Windows. Es gibt zwei Gründe dafür. Der eine ist die Hardware, der andere das spezielle Multitasking von Windows. Der Hardwaregrund ist einfach die Beschränkung des Arbeitsspeichers des 8086-Prozessors auf 1 Mbyte. Das Multitasking von Windows benötigt unbegrenzt viel Speicher, wenn immer mehr Anwendungen laufen. Dieses Problem ist in Windows teilweise durch eine virtuelle Speicherverwaltung gelöst. Nicht benötigte Code- und Ressourcensegmente sind als entfernbar (discardable) markiert, und können nach einem »least recently used«-Algorithmus auf Anforderung freigegeben werden. Natürlich liest Windows diese, falls sie wieder benötigt werden, vom Massenspeicher (Code- und Ressourcensegmente sind nur lesbar, was bedeutet, daß der Inhalt auf dem Massenspeicher sicherlich korrekt ist). Das Speicherproblem läßt sich durch Entfernen nicht vollständig lösen. Jede Anwendung hat einen minimalen Speicherbedarf, so daß zwei oder mehr große Programme nicht gleichzeitig laufen können. Dies ist ein Problem, da Anwender große Anwendungen am liebsten gleichzeitig laufen lassen, um Daten auszutauschen, DDE-Konversationen zu beginnen und um die Fähigkeiten von Windows zu nutzen, mehrere Programme gleichzeitig laufen zu lassen.

Windows 2.0 löst dieses Problem mit seinem verbesserten Memory Manager, der jeder Anwendung seinen eigenen Speicher im Expanded Memory gibt, wodurch die Anwendung eigentlich allein auf der Maschine läuft. Dies funktioniert immer, wenn eine EMS-Karte und ein EMS 4.0-Treiber vorhanden sind. Das Zwischenspeichern von Programm und Daten ist für Windows vollständig transparent. Der Windows-Speichermanager erledigt alle Aufrufe des EMM, um den Speicher zu allokieren, einzublenden, auszublenden und den Speicher wieder freizugeben, wenn die Anwendung beendet wird.

Die Speicherverwaltung könnte verbessert werden, um den EMS-Speicher so zu verwalten, daß einer einzigen Anwendung die ganzen 32 Mbyte der EMS-4.0-Unterstützung gegeben werden können. Die gegenwärtige Art wurde aus Implementationsgesichtspunkten, Geschwindigkeitsgründen und der Verfügbarkeit anderer Lösungen gewählt.

Eine Überlegung war, daß Windows 2.0 aufwärtskompatibel zu Windows 1.x sein muß. Das bedeutet, daß die Schnittstelle zur Speicherverwaltung gleich sein muß. Zum Benutzen des EMS-Speichers wäre eine neue Schnittstelle notwendig. Diese Schnittstelle existiert aber schon in der Form der EMS-4.0-Spezifikation. Wenn also eine Anwendung mehrere Mbyte an Speicher benötigt, kann sie diesen direkt von der EMS-Speicherverwaltung bekommen.

Der zweite Grund für den gegenwärtige Entwurf ist die notwendige Geschwindigkeit bei der Unterstützung von Expanded Memory: dem gleichzeitigen Ablauf von mehreren großen Windows-Anwendungen mit einem minimalen Aufwand beim Kontextumschalten. Die derzeitige Implementierung verlangt nur, daß jede Anwendung mit 640 Kbyte in einer akzeptablen Geschwindigkeit arbeiten kann. Es besitzt aber nicht jeder Computer Expanded Memory, und so kann ein Entwickler von Windows-Anwendungen diesen nicht voraussetzen. Nur beim Vertrieb einer Komplettlösung sieht dies anders aus.

Der dritte Grund, weswegen Windows EMS-Unterstützung nicht Mbyte von Expanded Memory für Anwendungen bietet, ist, daß ein Cache-Speicher das gleiche erledigt. Programm, Ressourcen und andere entfernbare Segmente können freigegeben und schnell mit der Hilfe der Möglichkeiten der Speicherverwaltung vom Disk Cache nachgeladen werden. Windows 2.0 beinhaltet ein Cache-Programm, das dynamisch EMS-Seiten allokiert und freigibt, ganz wie es gerade erforderlich ist. Im Gegensatz zu einer RAM-Disk ist die optimale Geschwindigkeit nicht von einer Benutzerentscheidung abhängig.

Windows 2.0 läuft im Gegensatz zu früheren Versionen in der Kompatibilitätsbox von OS/2. Sogar in dieser Betriebsart benutzt Windows Expanded Memory.

Arbeitsweise der EMS-Unterstützung

Die Speicherverwaltung von Windows ist sowohl für die Kontrolle der dynamischen Allokierung des Speichers vom globalen Heap als auch für das Allokieren vom lokalen Heap für jede Task oder Bibliothek zuständig. Hier interessiert uns nur die Allokierung vom globalen Heap. Die Speicherverwaltung von Windows bearbeitet Blockanfragen, indem sie die ganze Verwaltung erledigt wenn Blöcke verschoben oder entfernt werden müssen. Wenn ein Speicherblock wieder freigegeben wird, das heißt dem Speicherpool wieder zur Verfügung steht, merkt sich dies die Speicherverwaltung.

Windows blendet für jede Anwendung beim Laden eine neue Seite des EMS-Speichers ein. *Bild 3* zeigt eine Speicherbelegung mit drei Windows-Anwendungen: Write, PageMaker und Microsoft Excel, jede in ihrem eigenen Bereich von Expanded Memory-Seiten. Im Bild 3 sind gerade alle Seiten von Microsoft Excel in den physischen Adreßbereich eingeblendet. Beachten Sie, daß nur die minimal benötigte Anzahl von logischen Expanded Memory-Seiten für eine Anwendung allokiert wird, wodurch das Expanded Memory sehr effizient genutzt werden kann.

Bild 3 zeigt, daß alle Seiten von Microsoft Excel eingeblendet sind. Dies ist immer der Fall, wenn Microsoft Excel arbeitet, oder um bei der Terminologie von Windows zu bleiben: immer wenn Microsoft Excel eine Nachricht durch den Funktionsaufruf von `GetMessage` erhält. Klickt der Anwender das Fenster von PageMaker an, so wird durch eine Kontextumschaltung PageMaker zur aktiven Anwendung. Während der Kontextumschaltung blendet die Speicherverwaltung alle Seiten des Expanded Memory vom PageMaker ein, was bedeutet, daß alle Seiten von Microsoft Excel ausgeblendet werden. Das hat Auswirkungen auf den gemeinsamen Speicherzugriff von mehreren Anwendungen, was später erläutert wird.

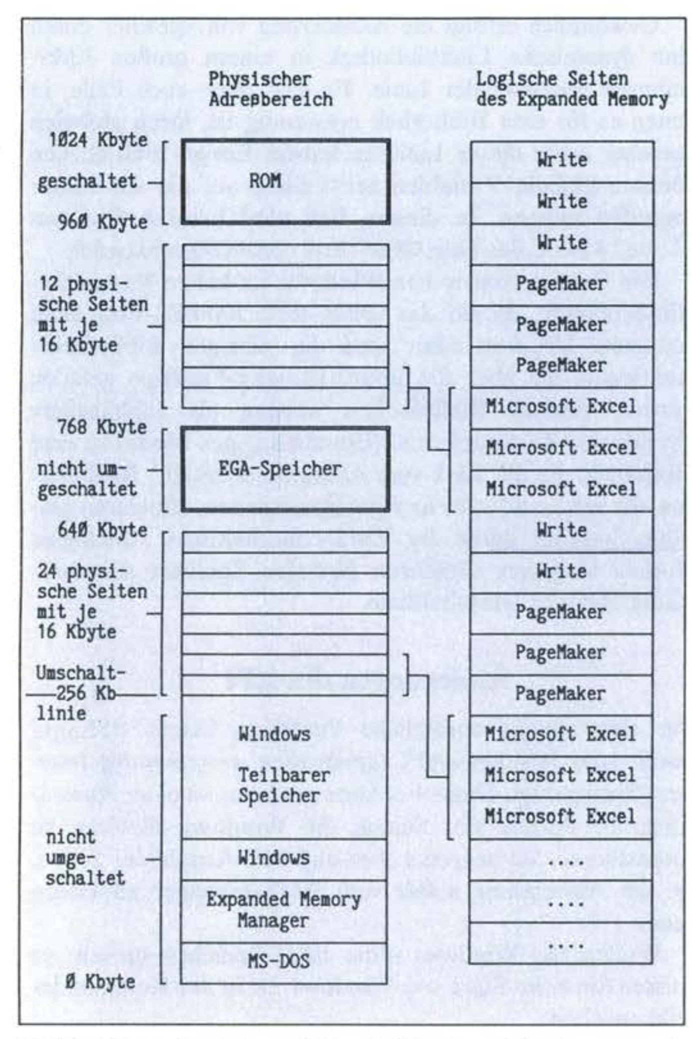


Bild 3: *Dieses Speicherabbild zeigt Code und Datensegmente der Programme Write, PageMaker und Microsoft Excel im Expanded Memory. Die Segmente von Microsoft Excel sind gerade in den physischen Adreßbereich der CPU eingeblendet.*

Während der Abarbeitung der Anwendung bleibt die Allokierung der Speicherbereiche fest. Das bedeutet, daß die Windows-Anwendungen in 640 Kbyte (in Wirklichkeit bis zu 256 Kbyte mehr, oder 896 Kbyte mit einer 64-Kbyte-EGA-Karte) laufen, und so programmiert sein sollten, daß sie innerhalb dieser Grenzen ein gutes Laufzeitverhalten zeigen.

Die einzigen Objekte, die entfernt werden, sind diejenigen, die im physischen Adreßbereich liegen, das heißt, in einer der gerade eingeblendeten EMS-Seite oder unterhalb der Umschaltlinie. Viele Objekte, die ausgeblendet sind, werden nie entfernt.

Speicherschutz

Ist der EMS-Speicher einer Anwendung nicht in den physischen Speicherbereich eingeblendet, so ist er geschützt gegen »unsaubere Anwendungen«. Wenn Windows mit großen EMS-Rahmen läuft, wenn also Expanded Memory in jeden Teil des physischen Adreßbereichs eingeblendet werden kann, dann ist jede Anwendung vollständig von den anderen geschützt. Stehen nur kleine Rahmen zur Verfügung, dann sind nur diejenigen Programmteile, die im Expanded Memory stehen, geschützt. Obwohl dieser Speicherschutz nicht so gut wie der von OS/2 ist, bei dem unerlaubte Adressierung zu einer generellen Schutzverletzung führt, so gehen diese Schutzfähigkeiten doch weit über diejenigen von Windows 1.x hinaus.

Speicherverwalter

Wenn Windows mit EMS-Speicher arbeitet, dann sind in Wirklichkeit zwei Speicherverwaltungen anwesend. Die Speicherverwaltung von Windows ist unter Windows 2.0 die primäre, sie ruft den EMM nur zur Kontrolle über das Expanded Memory auf. Seine Aufgabe ist das Allokieren, Ein- und Ausblenden sowie Freigeben des Expanded Memory. Ebenso muß er über die Register des EMS Buch führen. Da zwei Speicherverwaltungen vorhanden sind, kann eine Windows-Anwendung den EMM direkt zum Allokieren von Speicher ansprechen, wenn sie weiß, daß er vorhanden ist.

Eine Sache, die Sie sicher gerne wissen möchten, ist, welche Objekte über der Umschaltlinie angelegt werden, und daher ausgeblendet werden können, und welche unterhalb dieser Linie liegen, und daher von mehreren benutzt werden können. Diese Diskussion ist schwierig, da EMS 4.0 zwei Arten von Seitenrahmen unterstützt: kleine und große. Der kleine EMS-Seitenrahmen ist 64 Kbyte groß, der große hingegen kann bis zu 896 Kbyte groß sein - falls die Hardware es zuläßt. Da normalerweise der ganze Adreßbereich über 256 Kbyte bei der Verwendung eines großen Seitenrahmens umgeschaltet werden kann, bietet diese Lösung die meisten Möglichkeiten für die Speicherverwaltung von Windows. *Tabelle 1* zeigt, wie Expanded Memory bei beiden Typen von Seitenrahmen verwendet wird.

	Immer unterhalb der Linie	oberhalb der Linie	
		kleiner Rahmen	großer Rahmen
Datenbanken der Anwendung	√		
Modulheader (EXE)	√		
Bibliotheks-Datensegmente	√		
Fester Code in Bibliotheken	√		
Thunks	√		
Bibliotheksressourcen	√		
Codesegment der Anwendung		√	√
Ressourcen der Anwendung		√	√
Datensegmente der Anwendung			√
Entfernbarer Bibliothekscode			√
Dynamisch allokierter Speicher (über GlobalAlloc)			√

Tabelle 1: *Expanded Memory wird für beide Arten von Seitenrahmen benutzt.*

Kommt eine Anwendung mehr als einmal vor, dann verwendet die gegenwärtige Implementierung eine Bank gemeinsam (obwohl jedes Vorkommen sein eigenes Datensegment besitzt) und vermeidet so alle Probleme mit dem gemeinsamen Speicher, der entweder explizit durch `GetInstanceData` oder implizit durch gemeinsame Speicherhandles verwaltet wird. Spätere Implementierungen könnten dem Entwickler die Option bieten, für jedes Vorkommen einer Anwendung eine eigene EMS-Bank anzufordern.

Sicher gibt ein großer Rahmen der Speicherverwaltung die größtmögliche Flexibilität, aber sogar mit einem kleinen Rahmen ergibt die EMS-Unterstützung große Vorteile. Bei kleinen Anwendungen paßt der gesamte Programm- und Ressourcenbereich in den eingeblendeten Speicher, und gestattet auf diese Weise vielen kleinen Anwendungen ohne Geschwindigkeitsverlust zu arbeiten. Anwendungen wie die Systemsteuerung, der Taschenrechner und der Kalender können mit einer großen Anwendung wie etwa Microsoft Excel zusammen und ohne große gegenseitige Beeinträchtigung betrieben werden. Hier erlaubt sogar der kleine Rahmen eine bessere Ausnutzung der Multitasking-Fähigkeiten von Windows.

Dynamische Linkbibliotheken

Werden kleine Rahmen verwendet, werden nur die Ressourcen und die Programmteile einer Task in den EMS-Speicher geladen. Hingegen legt die Speicherverwaltung bei großen EMS-Rahmen viele andere Objekte oberhalb der Umschaltlinie an. Dies beinhaltet Datensegmente der Task, entfernbare Bibliotheks-Programmteile und Speicher, der mit `GlobalAlloc` allokiert wurde. Die letzten beiden müssen mit Sorgfalt behandelt werden, da sich dadurch einige Implementierungsprobleme ergeben.

Dynamische Bibliotheken können von jeder Task angesprochen werden. Dynamische Bibliotheken unter Windows sind zum Beispiel die Module Kernel, User und GDI, sowie alle Einheitentreiber (Tastatur, Maus, Timer, Bildschirm und Drucker). Es ist verständlich, daß Funktionen wie die GDI-Routine `TextOut` für jede Task ständig verfügbar sein müssen. Ist ein großer EMS-Rahmen verfügbar, so werden die Programmteile auf eine der beiden folgenden Weisen gehandhabt: Feste Programmteile liegen unter der Linie, entfernbare Programmteile liegen über der Linie.

Bei der Unterstützung großer EMS-Rahmen werden entfernbare Segmente der Bibliotheken über die Umschaltlinie der Task gelegt, wodurch die Wahrscheinlichkeit steigt, daß ein Teil der Bibliothek zu jeder Zeit in mehreren Banks erscheint. Zum Beispiel können mehrere Kopien des Dialogbox-Managers, der die Fensterprozeduren für die Edit-, Pushbutton- und andere Dialogbox-Steuerungen enthält, im EMS-Speicher vorhanden sein. Obwohl dies Redundanz bedeutet, haben die Anwendungen doch Zugriff auf die Funktionen, die sie benötigen. Außerdem wird wenig Platz im Speicher unter der Linie verbraucht.

Gewöhnlich erfolgt die Allokierung von Speicher durch eine dynamische Linkbibliothek in einem großen EMS-Rahmen oberhalb der Linie. Es gibt aber auch Fälle, in denen es für eine Bibliothek notwendig ist, ihren globalen Speicher unter dieser Linie zu haben. Einige Bibliotheken könnten globale Variablen verwenden, auf die sie immer zugreifen müssen. In diesem Fall wird beim Aufruf von `GlobalAlloc` das Flag `GMEM_NOT_BANKED` verwendet.

Die Druckertreiber von Windows 1.x haben Kompatibilitätsprobleme, da sie das `GMEM_NOT_BANKED`-Flag nicht benutzen. Druckertreiber sind die einzigen Bibliotheksfunktionen, die über die `LoadLibrary`-Funktion geladen werden. Andere Bibliotheken werden als »abhängige« Module von Tasks geladen. (Ein abhängiges Modul ist eine Bibliothek, die die Task zum Ablaufen benötigt.) Bibliotheken, die mit `LoadLibrary` geladen werden, allokieren globalen Speicher unter der EMS-Umschaltlinie. Abhängige Module hingegen allokieren globalen Speicher standardmäßig über der Umschaltlinie.

Änderungen des API

Nur eine einzige zusätzliche Funktion, `LimitEMSPage`, wurde zum Windows-API (*application programming interface*) hinzugefügt. Diese Funktion gestattet es einer Anwendung, die Anzahl der Seiten, die Windows allokiert, zu kontrollieren. Sie begrenzt aber nicht die Anzahl der Seiten, die die Anwendung selbst vom EMS-Manager allokieren kann.

Wollen Sie Windows ohne EMS-Speicher starten, so müssen Sie beim Start von Windows `/n` in der Kommandozeile angeben:

```
c>win /n
```

Virtueller Speicher

Der größte Vorteil von Windows/386 ist die Unterstützung von Standard-MS-DOS-Anwendungen. Die augenblickliche Version von Windows/386 erzeugt mehrere virtuelle Maschinen (VM), die sich alle wie unabhängige 8086-Rechner verhalten. Da jede seinen eigenen 8086 und 640 Kbyte Speicher zugeteilt bekommt, laufen die Anwendungen ohne sich gegenseitig zu beeinflussen. Da der 80386 die Möglichkeit besitzt, den direkten Bildschirmspeicherzugriff abzufangen, kann jede Anwendung in ihrem eigenen Fenster auf dem Bildschirm laufen.

Bedenken Sie aber, daß Windows/386 nicht jeder Windows-Anwendung seinen eigenen Speicher von 640 Kbyte gibt. Statt dessen laufen alle Windows-Anwendungen in zusammen 640 Kbyte Speicher. Dies bedeutet, daß die Windows-Anwendungen unter Windows/386 nicht dieselben Speichervorteile haben, wie MS-DOS-Anwendungen.

Die Unterstützung der Speicherumschaltung ist in Windows/386 dieselbe wie die EMS-Unterstützung in Windows 2.0. Sie arbeitet allerdings ein wenig anders. Da der 386

hardwaremäßige Unterstützung für die Speicherverwaltung bietet, sind die Unterschiede für die Windows-Anwendung transparent.

Programmierrichtlinien

Die hier dargestellten Programmierrichtlinien werden für einen Windows-Programmierer keine Überraschungen bieten. Sie sollen hier aber wiederholt werden, da die EMS-Unterstützung unter Windows 2.0 und die Unterstützung von virtuellem Speicher unter Windows/386 verlangen, daß diese Richtlinien genau eingehalten werden. Die Richtlinien gelten für beide Versionen, aber einige sind speziell für die EMS-Unterstützung bestimmt. Sie geben Ihnen die Gewißheit, daß Ihre Anwendung sowohl unter Windows 2.0 als auch unter Windows/386 läuft.

Effiziente Speicherausnutzung

Es gibt derzeit keine Pläne, eine Windows-Anwendung mit Hilfe der Windows-Speicherverwaltung Speicher über dem physischen Adreßraum von 1 Mbyte des 8086 allokieren zu lassen. Statt dessen führt Windows ein Einblenden von Speicher aus, so daß jede Anwendung glaubt, den ganzen Computer für sich zu haben.

Die Speicherverwaltung von Windows gestattet Ihnen zur Laufzeit die Angabe, daß global allokierter Speicher unter der Umschaltlinie allokiert wird, wodurch er für die Anwendungen immer verfügbar ist. Sie erreichen dies durch das Flag `GMEM_NOT_BANKED` beim Aufruf der Funktion `GlobalAlloc`. Sie sollten dies als letzten Ausweg wählen, da der Speicher unter der Umschaltlinie sehr kostbar ist und deshalb von den wichtigsten Systemobjekten benötigt wird. Wenn Sie keinen ungewöhnlichen Grund haben, sollten Sie diese Art der Allokierung nicht nutzen. Ein gutes Beispiel für eine globale Allokierung unter der Umschaltlinie ist der Speicher, der von einem Drucker benötigt wird.

Programmstruktur

Um die Laufzeit Ihrer Anwendung mit einem kleinen EMS-Rahmen zu optimieren, sollten Sie die wichtigsten Segmente als PRELOAD in der Moduldefinition (.DEF) kennzeichnen und ihre Namen an den Beginn der SEGMENTS-Liste schreiben. Die Geschwindigkeit wird erhöht, da Windows beim Laden einer Anwendung mit dem umschaltbaren EMS-Speicher beginnt. Erst nachdem der umschaltbare EMS-Speicher gefüllt ist, verwendet Windows bei kleinen Rahmen Speicher unterhalb der Umschaltlinie. Die Geschwindigkeit erhöht sich, da die Speicherverwaltung von Windows keine Objekte oberhalb der Umschaltlinie entfernt.

Gemeinsame Daten

Die Zwischenablage (Clipboard), DDE und Bibliotheken sind Arten des Datenaustauschs, die in den aktuellen und zukünftigen Versionen von Windows arbeiten. Anders als beim Clipboard oder DDE, sollten Sie Speicher nicht durch die Übergabe globaler Speicherhandles unter den Anwendungen austauschen. Die Übergabe von Long-Zeigern auf Datenobjekte sollte ebenso vermieden werden. Sie können sich nicht sicher sein, daß die benötigten Daten eingeblendet sind, wenn die darauf zugreifende Anwendung läuft. Anwendungen die diese Richtlinien ignorieren, werden unter zukünftigen Versionen von Windows 2.0 und Windows/386 nicht mehr laufen.

Vergewissern Sie sich, daß Sie die Daten vom Clipboard in Ihren eigenen Datenbereich kopieren, bevor Sie das Clipboard schließen. Windows unterstützt EMS, indem es die Objekte der Zwischenablage, die ausgeblendet wurden, in den gerade eingeblendeten Speicher kopiert. Nach dem Schließen der Zwischenablage werden diese Objekte gelöscht, was sie für die darauf zugreifende Anwendung ungültig macht.

Wenn eine Anwendung eine globale Handle von der Zwischenablage oder dem DDE erhält, muß es den Ergebniswert der Funktion `GlobalAlloc` prüfen. Die Windows-Speicherverwaltung kopiert in die aktuelle Seite diejenigen Zwischenablage- oder DDE-Objekte, die im ausgeblendeten EMS-Speicher liegen. Ist die Speicherverwaltung nicht in der Lage den Kopiervorgang durchzuführen (wenn zum Beispiel der Speicher zu knapp wird), informiert sie die Anwendung durch die Rückgabe von NULL beim Aufruf von `GlobalAlloc`.

Gemeinsamer Programmbereich

Eine Neuerung von Windows ist die Möglichkeit, Programmteile gemeinsam zu benutzen. Nur *eine* Kopie einer Windows-Funktion ist zu einer Zeit im Speicher. Sie kann durch einen dynamischen Linkmechanismus von allen Anwendungen aufgerufen werden. Wenn sie saubere Windows-Anwendungen programmieren, wird der gemeinsame Zugriff auf Programmteile durch den EMS-Speicher nicht beeinträchtigt. Da mehrere Vorkommen der gleichen Anwendung in denselben Seiten des EMS laufen, gibt es kein Problem mit dem gemeinsamen Zugriff auf Programm, Daten oder Ressourcen.

Dynamische Linkbibliotheken sind eine Möglichkeit zur gemeinsamen Verwendung von Programmteilen durch mehrere Anwendungen. Windows lädt die entfernbaren Bibliothekssegmente oberhalb der Umschaltlinie, so daß mehrere Kopien desselben Codesegments im EMS-Speicher vorkommen können. Diese Redundanz kann durch Konvertieren dieser Bibliothek in eine fensterlose Task und durch die Kommunikation mit den Message-Routinen `PostMessage` und `PostAppMessage` vermieden werden.

Sie können keine Programmteile, die zu einer anderen Task gehören, direkt aufrufen. Sie können also nie `GetProcAdress` aufrufen, um Programmteile einer anderen Task selbst aufrufen zu können. Obwohl dies in der Version 1.x funktionierte, läßt es die Speicherverwaltung von Windows 2.0 nicht zu. Statt dessen können Sie die Funktion

`SendMessage` für die Kommunikation zwischen verschiedenen Tasks benutzen.

Eine Anwendung kann den EMM direkt aufrufen, um EMS-Speicher zu allokieren, vorausgesetzt sie befolgt die Richtlinien der Version 3.2. Sie kann auch die Wiederallokierfunktion (ReAlloc: Funktion 17) von LIM EMS 4.0 verwenden. Windows muß dies mitgeteilt werden, so daß es nicht den 64 Kbyte großen Rahmen von EMS 3.2 verwendet. Dies geschieht durch die Angabe von `-LIM32` beim Aufruf des Ressourcen-Compilers. Windows vermeidet das Einblenden von Programmteilen in den 64 Kbyte großen EMS-3.2-Rahmen für Ihre Anwendung, es kann diesen Rahmen aber weiterhin für andere Anwendungen verwenden.

Globaler Speicher

Sie sollten keinen globalen Speicher unterhalb der Umschaltlinie allokieren. Sie sollten also das Flag `GMEM_NOT_BANKED` beim Aufruf von `GlobalAlloc` vermeiden. Wenn globale Objekte zwischen Anwendungen ausgetauscht werden, so sorgt der Aufruf von `GlobalLock` für das Kopieren der benötigten Blöcke. Dies sollte aber nur mit den Methoden der Zwischenablage und des DDE genutzt werden, um Kompatibilität mit neueren Versionen von Windows 2.0 und Windows/386 zu gewährleisten.

Ihr Programm sollte immer Fehler beim Allokieren, Wiederallokieren und beim Sperren von Speicher prüfen. Dies ist kritisch bei der Verwendung von DDE oder dem gemeinsamen Datenbereich der Zwischenablage, und es stellt eine gute Windows-Programmierpraxis dar. Ein ungültiger Far-Zeiger kann tödlich sein, da Daten irgendwo im Speicher geschrieben oder gelesen werden können.

Windows-Hooks

Die Windows-Anknüpfungspunkte (Hooks) erlauben Ihnen das Abfangen einiger Ereignisse vor dem Einfügen in die Systemschlange. Sie können zum Beispiel eine Tastaturüberwachung programmieren, die die Taste [Alt][P] überwacht. So kann ein Hintergrundprogramm einen Ausdruck des Bildschirms auf dem aktuellen Drucker durchführen.

Die Auswirkung des EMS-Speichers auf die Windows-Anknüpfungspunkte erfordert ein wenig zusätzlichen Programmieraufwand. Die Windows-Anknüpfungspunkte müssen unterhalb der Umschaltlinie in festen Bibliothekssegmenten liegen. Da der Windows-Anknüpfungspunkt nie weiß, ob die Task gerade eingeblendet ist, muß er mit der aufrufenden Task über eine Nachricht kommunizieren (zum Beispiel `SendMessage`).

Debugger-Unterstützung

Sie können jetzt zum Fehlersuchen in einer Windows-2.0-Anwendung CodeView benutzen. Das stellt eine wesentliche Verbesserung gegenüber Symdeb - einem mächtigen, aber kryptischen Debugger - dar, der am besten für Assembler-Programmierer geeignet ist. CodeView verlangt entweder einen monochromen Monitor oder ein Terminal, da Windows die Kontrolle über den Bildschirm übernimmt. Eine weitere Voraussetzung ist das Vorhandensein von EMS-Speicher, da CodeView direkt die Expanded Memory-Speicherverwaltung anspricht, um den Speicher, der der Anwendung fehlt, möglichst gering zu halten.

Um das Debuggen zu erleichtern, wenn EMS vorhanden ist, sollten Sie die folgenden Zeilen in die Datei WIN.INI schreiben:

```
[kernel]
EnableEMSDebug=1
```

Dies erlaubt Ihnen das Setzen von Breakpunkten in entfernbaren Bibliotheksfunktionen. Erinnern Sie sich, daß es mehrere Kopien von diesen entfernbaren Bibliotheksfunktionen in verschiedenen EMS-Seiten geben kann. Der Schalter teilt der Speicherverwaltung mit, daß die Tabelle der entfernbaren Bibliotheksfunktionen bei der Kontextumschaltung des Debuggers aktualisiert werden muß. Jeder Debugger, der mit Windows zusammenarbeitet wie Symdeb, CodeView, Answer oder Atron arbeitet auch richtig mit EMS.

Zusammenfassung

In den meisten Fällen ist die Expanded Memory-Speicherunterstützung von Windows für den Programmierer transparent. Mit der Ausnahme einer neuen Windows-Bibliotheksfunktion, `LimitEMSPages`, einigen neuen Flags und einer neuen Option des Ressourcen-Compilers, hat sich die Schnittstelle, die Windows-Programmierer verwenden, nicht geändert. Was sich aber geändert hat, ist die strengere Trennung zwischen den Tasks. Der Zugriff auf gemeinsame Daten muß nach einer der drei Arten erfolgen: Zwischenablage, DDE oder über gemeinsame dynamische Linkbibliotheken. Der ohnehin problematische direkte Zugriff auf Funktionen einer anderen Task, ist unter der Speicherverwaltung von Windows 2.0 unmöglich.

Programmierer haben den größten Vorteil vom Geschwindikeitszuwachs unter Windows 2.0, wenn Sie viele große Anwendungen laufen lassen. Dies läßt Produkte, die mit Anwendungen wie Microsoft Excel oder PageMaker zusammenarbeiten, attraktiver erscheinen. Windows-Entwickler können natürlich auch Ihre eigenen leistungsfähigen und großen Anwendungen programmieren. Da die EMS-Unterstützung für den Programmierer transparent ist, kann er die bestehende Programmschnittstelle verwenden, und die Windows-Speicherverwaltung die Vorteile von EMS nutzen lassen.

Paul Yao

Fachbegriffe der Windows-Speicherspeicherverwaltung

Umschaltlinie (bank line): Auch Grenze der EMS-Auslagerung genannt. Dies ist die Grenze, unter der der Speicher nicht umgeschaltet wird, und über der er umgeschaltet wird. Wenn Windows mit Expanded Memory arbeitet, ist der nicht umgeschaltete Speicher immer verfügbar, der umgeschaltete aber nur, wenn die Task läuft, der dieser Speicher gehört.

Umgeschalteter Speicher (banked memory): Speicher, der innerhalb des physischen Adreßbereichs des 8088 liegt (eingeblendet ist), wohin er von einem Treiber (Expanded Memory Manager) in Verbindung mit einer Expanded Memory-Karte gelegt wird.

Kontextumschaltung: Die Änderungen, die Windows durchführen muß, wenn der Anwender von einer Applikation zu einer anderen wechselt. Es wird zum Beispiel sowohl Windows Write als auch Windows Paint geöffnet. Wenn der Anwender Text in Windows Write eingegeben hat, und die Maus in Windows Paint anklickt, vollzieht das System eine Kontextumschaltung. Die einzige Auswirkung für den Anwender ist die Farbänderung in der Menüleiste. Wurde das jetzt aktive Fenster von einer anderen Anwendung verdeckt, holt Windows 2.0 es an die Oberfläche.

Entfernbare Speicherobjekte (Discardable memory object): Eines der Flags, die die Speicherverwaltung setzt, um die verschiedenen Arten von Speicherobjekten beim Allokieren von Speicher zu kennzeichnen. Kann die Windows-Speicherverwaltung eine Allokierung durch einfaches Verschieben von Speicherobjekten nicht erfüllen, so gibt sie entfernbare Speicherobjekte nach einem »least recently used«-Verfahren frei.

Dynamisch allokierter Speicher: Der Speicher, den ein Windows-Programm direkt von der Windows-Speicherverwaltung anfordert. In diesem Artikel ist dies gleichbedeutend mit global allokiertem Speicher.

Enhanced Expanded Memory Specification (EEMS): Eine Erweiterung der Expanded Memory-Spezifikation, die es gestattet, jeden Teil des Adreßbereichs des 8088 ein- und auszublenden.

Expanded Memory Specification (EMS): Diese Spezifikation, die auch unter dem Namen LIM EMS bekannt ist, definiert, wie ein Programm Expanded Memory allokieren und benutzen muß. Dies beruht auf Aufrufen des Expanded Memory-Einheitentreibers.

Expanded Memory Manager (EMM): Ein Einheitentreiber (device driver), der Expanded Memory kontrolliert. EMS definiert, wie ein Programm mit dem EMM kommuniziert, um Expanded Memory zu allokieren, einzublenden, und wieder freizugeben.

Extended Memory: Der Speicher über 1 Mbyte bei einem 80286- oder 80386-Computer. Diese Art von Speicher kann nur im Protected Modus direkt adressiert werden, so wie OS/2 dies tut. Normalerweise wird Extended Memory für RAM-Disks oder Druckerspooler benutzt. Viele Speicherkarten können für Expanded, Extended oder gemischte Adressierung eingestellt werden.

Handle: Ein Wert, der beim Erzeugen eines Objekts zurückgegeben wird. Handles identifizieren von Windows dynamisch allokierte Speicherobjekte.

Logische Seite: Ein Segment des Expanded Memory. EMS-Speicher wird in logischen Seiten allokiert, die gewöhnlich 16 Kbyte groß sind. Um auf Daten in diesen logischen Seiten zugreifen zu können, muß EMS die logischen Seiten in den physischen Rahmen einblenden.

Einblenden: Die Art, wie eine logische Seite des Expanded Memory in den Adreßbereich des 8088 gelegt wird, um sie vom Programm aus ansprechen zu können.

Unterstützung alter Applikationen: Die Fähigkeit von Windows, zeichenorientierte Standard-MS-DOS-Applikationen, wie Microsoft Word, Lotus 1-2-3 oder Microrim R:BASE System V auszuführen.

Seitenrahmen (page frame): Eine Ansammlung physischer Seiten, in die die logischen Seiten eingeblendet werden können.

Physische Seiten: Der absolute Adreßbereich im 1 Mbyte-Adreßraum des 8088, in den die logischen EMS-Seiten eingeblendet werden.

Thunk: Ein kleines Programmstück, das an einer festen Stelle im Speicher sitzt, um Aufrufe einer Funktion in einem verschiebbaren oder entfernbaren Segment abzufangen. Thunks sind ein wichtiger Teil der dynamischen Linkbibliotheken von Windows. Dies erlaubt der Speicherverwaltung von Windows das Verschieben oder Entfernen von Segmenten mit geringem Aufwand. Weitere Aufrufe veranlassen, das Segment mit der benötigten Routine nachzuladen. Return-Thunks werden erzeugt, um die Rückkehr zu einem entfernten Programmteil abzufangen. Thunks führen auch Kontextumschaltungen durch. Das der von `MakeProcInstance` erzeugte Thunk, um sicherzustellen, daß eine Dialogbox das richtige Datensegment benutzt.

Fenster: Ein anderes Wort für den Seitenrahmen von EMS.

Speicherverwaltung von Windows: Der Teil von Windows, der Programm- und Datensegmente in den Speicher lädt. Die Speicherverwaltung von Windows versucht, Anforderungen von dynamischem Speicher zuerst durch das Verschieben von Objekten zu erfüllen. Funktioniert dies nicht, dann entfernt sie Objekte, die als entfernbar gekennzeichnet sind. Die Speicherverwaltung von Windows 2.0 wurde um die Fähigkeit der Speicherumschaltung zwischen Windows-Anwendungen erweitert.

Das System Journal versucht seine Bücherflut zu bewältigen (Teil II):

Einige Kurzbesprechungen

Es sind in den letzten Wochen wieder so viele interessante Bücher in der Redaktion eingetroffen, daß es einfach nicht mehr möglich ist, sie alle zu lesen, geschweige denn ausführlich zu besprechen. Etliche sollen jedoch zumindest mit kurzer Inhaltsangabe und bibliographischen Informationen vorgestellt werden.

OS/2

Wer vor englischsprachiger Fachliteratur nicht zurückschreckt findet in David E. Cortesis Buch »Essential OS/2« auf 450 Seiten viele nützliche Informationen und Beispiele für die Programmierung unter OS/2 1.0. Auf zusätzlichen 230 Seiten werden detailliert alle Funktionen der Programmierschnittstelle beschrieben.

David E. Cortesi: »The Programmer's Essential OS/2 Handbook: Redwood City: M&T Publishing, 1988; 709 Seiten; ISBN 0-934375-82-8; $24.95. 3 Disketten $25.

Windows

In der letzten Ausgabe haben wir das erste deutsche Windows-Buch vorgestellt, doch dabei ist es nicht geblieben. Der Systhema Verlag bringt mit »Programmierung unter Windows« eine Einführung in die Windows-Programmierung für Windows-interessierte Umsteiger. Der Autor Tim Farrell schöpft aus seinen Erfahrungen bei der Entwicklung mehrerer kommerzieller Windows-Anwendungen und demonstriert am Beispiel eines einfachen Programmeditors Schritt für Schritt die für viele Programmierer neuen Aspekte der meldungsgesteuerten Programmierung unter Windows.

Tim Farrell: »Programmierung unter Windows«, München: Systhema Verlag, 1989; 500 Seiten; inklusive 2 Disketten; ISBN 3-89390-251-1; DM 98,-.

Programmiersprachen

Während C im professionellen Bereich ganz klar die favorisierte Programmiersprache ist, gebührt BASIC dieser Platz im Amateurlager und semiprofessionellen Bereich.

Für C bietet der Verlag McGraw-Hill die »C-Befehlsbibliothek«, ein umfassendes Nachschlagewerk zur Sprache C, an. Der bekannte C-Autor Herbert Schildt legt hiermit einen ausführlichen Überblick über die Sprache C und ihre Laufzeitbibliothek vor. Dabei wird auch die neueste Entwicklung (ANSI-Norm, C++) berücksichtigt.

Herbert Schildt: »C-Befehlsbibliothek«, Hamburg: McGraw-Hill, 1988; 752 Seiten; ISBN 3-89028-142-7.

H. Herold und W. Unger gaben ihrem Werk über C den Titel »C-Gesamtwerk«, wodurch man zunächst glaubt, daß es sich um ein ähnliches Nachschlagewerk, wie das Buch von H. Schildt handelt. Doch die Struktur diese Buchs ist ganz anders. Hier wird an über 100 kommentierten kleinen Programmbeispielen die Realisierung moderner Programmstrategien in C vorgeführt. Berücksichtigt werden dabei C-Compiler unter allen bekannten Betriebssystemen, von CP/M über DOS bis zu Unix.

Helmut Herold, Werner Unger: »C-Gesamtwerk«, München: te-wi Verlag, 1988; 570 Seiten; ISBN 3-89362-015-X; DM 79,-.

Um die Umsetzung eines Problems in ein lauffähiges BASIC-Programm geht es in H.-J. Sachts Einführung in das Programmieren mit BASIC. Dieses 1983 zum ersten Mal erschienene Werk wurde für die vorliegende 4. Auflage stark überarbeitet und erweitert. Es eignet sich vor allem für Programmiereinsteiger.

Hans-Joachim Sacht: »Vom Problem zum Programm/Programmieren in GW-BASIC, Turbo BASIC und QuickBASIC«, Würzburg: Vogel Verlag, 4. Aufl. 1988; 462 Seiten; ISBN 3-8023-0214-1; DM 48,-.

Wesentlich weiter führt »Das große PC-BASIC-Buch« von Heinz-Josef Bomanns. Hier werden in ausführlichen Beispielen praxisorientierte Themen wie Grafik, Sound, Dateiverwaltung, Bildschirmansteuerung, Druck usw. beschrieben. Auch die BASIC/Assembler-Kopplung kommt nicht zu kurz. Auf kompaktem Raum hat Bomanns sein Know-How zu QuickBASIC in einem Data Becker Führer zusammengefaßt. Es werden dabei alle Versionen von 2.0 bis 4.0 berücksichtigt.

Heinz-Josef Bomanns: »Das große PC-BASIC-Buch (GW-BASIC/QuickBASIC«, Düsseldorf: Data Becker, 1988; 734 Seiten; mit einer Diskette; ISBN 3-89011-240-4; DM 69,-.

Heinz-Josef Bomanns: »Der DATA BECKER Führer QuickBASIC«, Düsseldorf: Data Becker, 1988; 250 Seiten; ISBN 3-89011-451-2; DM 24,80.

Auch FORTRAN, der Oldtimer unter den Programmiersprachen, wird (vor allem bei rechenintensiven Anwendungen) noch immer eingesetzt. Das Buch beschreibt, wie man mit Microsoft FORTRAN 4.0 zeitgemäß strukturiert programmiert.

Hans Mahnke: »Strukturiert programmieren in Fortran 77«, Würzburg: Vogel Verlag, 1988; 126 Seiten; ISBN 3-8023-0221-4; DM 30,-.

Wer keine umfangreichen Beispiele für die Programmierung unter DOS benötigt und mit einer kurzen Auflistung der Programmierschnittstelle auskommt, dem sei folgendes kleine Büchlein empfohlen:

Ray Duncan: »MS-DOS Functions«, Redmond: Microsoft Press, 1988; 122 Seiten; ISBN 1-55615-128-4; $5.95.

»Profi-Tools QuickC«:

Stringbehandlung in C

In dem Buch »Profi-Tools QuickC« wird dem Leser eine Library mit etwa 60 Funktionen zur Verfügung gestellt, die ihm das Leben erleichtern sollen. Diese Library ist sowohl für die Benutzung mit dem QuickC- als auch mit dem C-5.0-Compiler geeignet. Wir bringen hier als Buchauszug daraus das Modul MYSTRING mit Routinen zur Stringbehandlung.

MYSTRING enthält in der Praxis häufig benötigte Stringfunktionen, die nicht in der QuickC-/C 5.0-Library enthalten sind. Zum Beispiel Funktionen zum Löschen oder Einfügen eines Strings, Funktionen zum Durchsuchen eines Strings nach dem nächsten beziehungsweise vorhergehenden Wort, oder Funktionen, mit denen es möglich ist, ohne wiederholten Aufruf von `concat` einen String aus mehreren Teilstrings zu bilden. MYSTRING enthält also Funktionen, die den Umgang mit Strings erheblich vereinfachen.

`charins`: ein Zeichen in einen String einfügen
`strins`: einen String in einen zweiten String einfügen
`strdel`: einen Teil eines Strings entfernen
`strfill`: String teilweise mit einem Zeichen füllen
`strstring`: einen String mit einem Zeichen füllen
`strpad`: einen String mit Leerzeichen auffüllen
`strcompress`: »überflüssige« Leerzeichen löschen
`lower`: Groß- in Kleinbuchstaben wandeln
`upper`: Klein- in Großbuchstaben wandeln
`instr`: das Enthaltensein eines Strings prüfen
`nextword`: das nächste Wort suchen
`prevword`: das vorhergehende Wort suchen
`stradd`: einen String aus mehreren Teilstrings bilden
`charadd`: einen String aus mehreren Zeichen bilden

Wenn Sie einen Blick in die Dokumentation Ihres Compilers werfen, kommen Ihnen `lower` und `upper` auf den ersten Blick wahrscheinlich überflüssig vor. Im Gegensatz zu den dort beschriebenen Makros berücksichtigen diese beiden Funktionen jedoch auch Umlaute korrekt.

Beachten Sie bitte bei allen Funktionen, die die Länge des Ausgangs-Strings verändern, daß für den String ausreichend Platz reserviert sein muß (`charins`, `strins`, `strstring`, `strpad`, `stradd`, `charadd`). Keine der Stringfunktionen allokiert Speicherplatz für den String; er wird in allen Fällen als bereits reserviert vorausgesetzt!

charins: Zeichen in String einfügen

Syntax: `void charins(char ch, char *s)`
Eingabeparameter: `ch`: einzufügendes Zeichen, `s`: Pointer auf Einfügeposition im String
Funktionswert: keiner
Funktion: `charins` fügt ein Zeichen in einen String ein und verschiebt alle nachfolgenden Zeichen um eine Position (und das abschließende `'\0'`). Die Länge des resultierenden Strings ist `strlen(s) + 1`.
Beispiel:

```
#include <stdio.h>
#include "myheader.h"
#include "myinit.h

main()
{
   char s[81];

   strcpy(s, "Des ist ein Test");
   charins('i', &s[1]);
   printf("%s", s);
}


=> Dies ist ein Test
```

strins: Einen String in einen zweiten einfügen

Syntax: `void strins(char *s1, char *s2)`
Eingabeparameter: s1: Pointer auf den einzufügenden String, s2: Pointer auf die Einfügeposition
Funktionswert: keiner
Funktion: `strins` fügt einen String `s1` in einen zweiten String `s2` ein; alle Zeichen, die der Einfügeposition folgen (inklusive `'\0'`), werden verschoben.
Beispiel:

```
#include <stdio.h>
#include "myheader.h"
#include "myinit.h"

main()
{
      char s[81];

   strcpy(s, "Dies ein Test");
   strins(" ist", &s[4]);
   printf("%s", s);
}


=> Dies ist ein Test
```

strdel: Teil eines Strings entfernen

Syntax: `void strdel(char *start, char *ende)`
Eingabeparameter: `start`: Pointer auf das erste zu entfernende Zeichen, `ende`: Pointer auf das letzte zu entfernende Zeichen

Funktionswert: keiner
Funktion: `strdel` entfernt alle Zeichen eines Strings von `start` bis `ende`; alle `ende` folgenden Zeichen (inklusive `'\0'`) werden nach `start` ff. verschoben.
Beispiel:

```
#include <<stdio.h>>
#include "myheader.h"
#include "myinit.h"

main()
{
   char s[81];

   strcpy(s, "Dies ist ist ein Test");
   strdel(&s[4], &s[7]);
   printf("%s", s);
}

=> Dies ist ein Test
```

strfill: String teilweise mit einem Zeichen füllen

Syntax: `void strfill(char *start, char *ende, char ch)`
Eingabeparameter: `start`: Pointer auf das erste zu überschreibende Zeichen, `ende`: Pointer auf das letzte zu überschreibende Zeichen, `ch`: Zeichen, mit dem überschrieben wird
Funktionswert: keiner
Funktion: `strfill` überschreibt alle Zeichen eines Strings von `start` bis `ende` mit dem Zeichen `ch`.
Beispiel:

```
#include <<stdio.h>>
#include "myheader.h"
#include "myinit.h"

main()
{
  char s[81];
  strcpy(s, "Seriennummer       , 2.5.88");
  strfill(&s[13], &s[18], 'X');
  printf("%s", s);
}
=> Seriennummer XXXXXX, 2.5.88
```

strstring: Einheitlichen String bilden

Syntax: `char *strstring(char *s, char ch, int len)`
Eingabeparameter: `s`: Pointer auf den String, `ch`: Zeichen, aus dem der String bestehen soll, `len`: gewünschte Stringlänge
Funktionswert: Pointer auf den String
Funktion: `strstring` erzeugt einen mit `'\0'` abgeschlossenen String `s` der Länge `len`, der mit dem Zeichen `ch` gefüllt ist.
Beispiel:

```
#include <stdio.h>
#include "myheader.h"
#include "myinit.h"

main()
{
   char s[81];

   strstring(s, 'X', 10);
   printf("%s", s);
}

=> XXXXXXXXXX
```

strpad: String mit Leerzeichen auffüllen

Syntax: `char *strpad(char *s, int len)`
Eingabeparameter: `s`: Pointer auf den String, `len`: gewünschte Stringlänge
Funktionswert: Pointer auf den String
Funktion: `strpad` hängt an das Ende eines gegebenen Strings Leerzeichen an, um ihn auf eine Sollänge aufzufüllen. Der String wird mit `'\0'` abgeschlossen.
Beispiel:

```
#include <stdio.h>
#include "myheader.h"
#include "myinit.h"

main()
{
   char s[81];

   strcpy(s, "Test");
   strpad(s, 10);
   strcat(s, "Test1");
   printf("%s", s);
}

=> Test      Test1
```

strcompress: Abschließende Leerzeichen entfernen

Syntax: `char *strcompress(char *s)`
Eingabeparameter: s: Pointer auf den String
Funktionswert: Pointer auf den String
Funktion: `strcompress` ist die Umkehrung von `strpad`; alle »überflüssigen« Leerzeichen am Ende eines Strings werden entfernt.
Beispiel:

```
#include <stdio.h>
#include "myheader.h"
#include "myinit.h"

main()
{
   char s[81];

   strcpy(s, "Test    ");
   strcompress(s);
   strcat(s, "ende");
   printf("%s", s);
}

=> Testende
```

lower: Großbuchstaben eines Strings in Kleinbuchstaben wandeln

Syntax: `char *lower(char *s)`
Eingabeparameter: s: Pointer auf den String
Funktionswert: Pointer auf den String
Funktion: `lower` wandelt alle in einem String enthaltenen Großbuchstaben in Kleinbuchstaben und berücksichtigt dabei auch Umlaute.
Beispiel:

```
#include <stdio.h>
#include "myheader.h"
#include "myinit.h"


main()
{
   char s[81];

   strcpy(s, "Dies ist ein Test");
   lower(s);
   printf("%s", s);
}

=> dies ist ein test
```

upper: Kleinbuchstaben eines Strings in Großbuchstaben wandeln

Syntax: `char *upper(char *s)`
Eingabeparameter: s: Pointer auf den String
Funktionswert: Pointer auf den String
Funktion: `upper` wandelt alle in einem String enthaltenen Kleinbuchstaben in Großbuchstaben und berücksichtigt dabei auch Umlaute.
Beispiel:

```
#include <stdio.h>
#include "myheader.h"
#include "myinit.h"

main()
{
   char s[81];

   strcpy(s, "Dies ist ein Test");
   upper(s);
   printf("%s", s);
}

=> DIES IST EIN TEST
```

instr: Enthaltensein eines Strings prüfen

Syntax: `char *instr(char *s1, char *s2)`
Eingabeparameter: `s1`: Pointer auf zu suchenden Teilstring, `s2`: Pointer auf den zu durchsuchenden String
Funktionswert: Pointer auf die gefundene Position (`NULL`, wenn `s1` nicht in `s2` enthalten ist
Funktion: `instr` prüft, ob der String `s1` im String `s2` enthalten ist. Wenn ja, übergibt `instr` einen Pointer auf die gefundene Position; ist `s1` nicht in `s2` enthalten, wird `NULL` übergeben.
Beispiel:

```
#include <stdio.h>
#include "myheader.h"
#include "myinit.h"

main()
{
   char s[81], *p;

   strcpy(s, "Dies ist ein Test");
   p = instr("ist", s);
   printf("%s", p);
}

=> ist ein Test
```

nextword: Nächstes Wort suchen

Syntax: `char *nextword(char *s, char *term)`
Eingabeparameter: `s`: Pointer auf den String, auf die »Startposition«, ab der die Suche beginnt, `term`: Pointer auf einen String mit den »Terminatoren« (Zeichen, die zwei Wörter voneinander trennen)
Funktionswert: Pointer auf den Anfang des nächsten Wortes (`NULL`, wenn kein nächstes Wort vorhanden ist)
Funktion: `nextword` sucht in einem String `s` den Beginn des nächsten Wortes. Wo ein »nächstes Wort« beginnt, bestimmt der String `term`. Alle darin enthaltenen Zeichen werden als Trennzeichen zwischen zwei Wörtern interpretiert. Wird ein Wort gefunden, übergibt `nextword` einen Zeiger darauf, sonst `NULL`.
Beispiel:

```
#include <stdio.h>
#include "myheader.h"
#include "myinit.h"

main()
{
   char s[81], *p;

   strcpy(s, "Hallo. Noch mal Hallo.");
   p = nextword(&s[2], "., !;()");
   printf("%s", p);
}


=> Noch mal Hallo.
```

prevword: Vorhergehendes Wort suchen

Syntax: `char *prevword(char *start, char *ende, char *term)`
Eingabeparameter: `start`: Pointer auf den Stringanfang, bis zu dem gesucht wird `ende`: Pointer auf das Ende des zu durchsuchenden Bereichs, `term`: Pointer auf einen String mit den »Terminatoren« (Zeichen, die zwei Wörter voneinander trennen)
Funktionswert: Pointer auf den Anfang des vorhergehenden Wortes (`NULL`, wenn kein vorhergehenden Wort vorhanden ist)
Funktion: `prevword` ist die Umkehrung von `nextword` und erfüllt die gleiche Aufgabe, durchsucht den String jedoch nach links! Die Suche beginnt bei `ende` und endet bei `start` (`start` muß immer kleiner sein als `ende`). `prevword` übergibt einen Pointer auf den Beginn des vorhergehenden Wortes beziehungsweise `NULL`, wenn die Suche erfolglos ist.
Beispiel:

```
#include <stdio.h>
#include "myheader.h"
#include "myinit.h"

main()
{
   char s[81], *p;

   strcpy(s, "Hallo. Noch mal Hallo.");
   p = prevword(s, s + strlen(s),
       "., !;()");
   printf("%s", p);
}


=> Hallo.
```

stradd: Strings verknüpfen

Syntax: `char *stradd(char *s, char *arg1, ..., NULL)`
Eingabeparameter: `s`: Pointer auf den »Summenstring«, `arg1`: Pointer auf den 1.Teilstring, `NULL`: Endekennzeichen
Funktionswert: Pointer auf den zusammengesetzen String
Funktion: `stradd` »addiert« eine beliebige Stringanzahl. Die angegebenen Strings werden mit `strcat` verkettet und zu einem String `s` verknüpft. Dem letzten Teilstring muß `NULL` folgen. Die resultierende Stringlänge entspricht der Summe der Länge der Teilstrings.
Beispiel:

```
#include <stdio.h>
#include "myheader.h"
#include "myinit.h"

main()
{
   char s[81];

   stradd(s, "Dies ", "ist ", "ein ",
             "Test", NULL);
   printf("%s", s);
}


=> Dies ist ein Test
```

charadd: Zeichen verknüpfen

Syntax: `char *charadd(char *s, char arg1, ..., '\0')`

Eingabeparameter: `s`: Pointer auf den »Summenstring«, `arg1`: das erste Zeichen, `'\0'`: Endekennzeichen

Funktionswert: Pointer auf den gebildeten String

Funktion: `charadd` verknüpft beliebig viele Zeichen zu einem mit `'\0'` abgeschlossenen String `s`. Dem letzten Zeichen muß als Endekennzeichen `'\0'` folgen. Die resultierende Stringlänge ist gleich der Anzahl der `char`-Argumente.

Beispiel:

```
#include <stdio.h>
#include "myheader.h"
#include "myinit.h"

main()
{
   char s[81];

   charadd(s, 'T', 'e', 's', 't', '\0');
   printf("%s", s);
}

=> Test
```

Said Baloui

Dieser Artikel ist ein Auszug aus dem Buch »Profi-Tools QuickC« von Said Baloui, erschienen im Markt&Technik Verlag, mit dessen freundlicher Genehmigung der Abdruck erfolgt.

```
/* STRING-Funktionen (MYSTRING.C) */
void charins              (char ch, char *s);
void strins               (char *s1, char *s2);
void strdel               (char *start, char *ende);
void strfill              (char *start, char *ende, char ch);
char *strstring           (char *s, char ch, int len);
char *strpad              (char *s, int len);
char *strcompress         (char *s);
char *lower               (char *s);
char *upper               (char *s);
char *instr               (char *s1, char *s2);
char *nextword            (char *s, char *term);
char *prevword            (char *start, char *ende, char *term);
char *stradd              (char *s, ...);
char *charadd             (char *s, ...);
```

Listing 1: `MYHEADER.H`.

```
/*   MYSTRING.C   */
/*   ==========   */
/*** "Standard"-Stringroutinen ***/

#include <stdio.h>
#include <string.h>
#include <ctype.h>

/*==========================================================

Die Funktionen CHARINS, STRINS und STRPAD erzeugen einen
String, der länger als der Ausgangsstring ist. Diese Funktionen
gehen davon aus, daß für den zu manipulierenden String
ausreichend Platz reserviert ist. Beachten Sie, daß auch
STRSTRING je nach angegebener Stringlänge einen String
erweitert und für STRADD ausreichend Platz für die Summe aller
verknüpften Strings zur Verfügung stehen muß.
*/
/*==========================================================


charins
-------
Funktion: Ein Zeichen <ch> in einen String <s> einfügen.
          Resultierende Länge: strlen(s) + 1.
Hin:      ch : Zeichen
          s  : Ptr. auf Einfügeposition im Ausgangsstring
*/
void charins(char ch, char *s)
{
   int i;

   for (i = strlen(s) + 1, s += i - 1; i; i--, s--)
      *(s + 1) = *s;
   *(++s) = ch;
}
/*==========================================================


strins
------
Funktion: String <s1> in String <s2> einfügen.
          Resultierende Länge: strlen(s1) + strlen(s2).
Hin:      s1 : Ptr. auf einzufügenden String
          s2 : Ptr. auf Einfügeposition im Ausgangsstring
*/
void strins(char *s1, char *s2)
{
   int i, len;

   for (len = strlen(s1), i = strlen(s2) + 1, s2 += i - 1;
        i; i--, s2--)
      *(s2 + len) = *s2;
   s2++;

   for ( ; *s1; s1++, s2++)
      *s2 = *s1;
}
/*==========================================================


strdel
------
Funktion: Entfernt aus einem String alle Zeichen von <start>
          bis <ende> durch Links-Verschieben der nachfolgenden
          Zeichen (inkl.'\0').
Hin:      start : Ptr. auf das erste zu löschende Zeichen
          ende  : Ptr. auf das letzte zu löschende Zeichen
*/
void strdel(char *start, char *ende)
{
   ende++;
   while (*start++ = *ende++)
      ;
}
/*==========================================================
```

Listing 2: `MYSTRING.C`.

```
strfill
-------
Funktion: Überschreibt in einem String alle Zeichen ab <start>
          bis <ende> mit dem Zeichen <ch>
Hin:      start : Ptr. auf das erste zu überschreibende Zeichen
          ende  : Ptr. auf das letzte zu überschreibende
                  Zeichen
          ch    : "Füllzeichen"
*/
void strfill(char *start, char *ende, char ch)
{
  int i;

  for (i = ende - start + 1; i; i--)
     *ende-- = ch;
}
/*============================================================

strstring
---------
Funktion: Erzeugt einen - mit '\Ø' abgeschlossenen - String
          der Länge <len>, gefüllt mit dem Zeichen <ch>
Hin:      s   : Ptr. auf String
          len : Sollänge des Strings
       RETURN : Ptr. auf erzeugten String
*/
char *strstring(char *s, char ch, int len)
{
  int i;
  char *p;

  p = s;
  for (i = Ø; i < len; i++)
     *s++ = ch;
  *s = '\Ø';
  return (p);
}
/*============================================================

strpad
------
Funktion: Füllt einen String bis zur angegebenen Länge mit
          Leerzeichen auf und schließt den String mit '\Ø'
          ab.
          Result. Länge: len.
Hin:      s      : Ptr. auf den String
          len    : Sollänge
         RETURN  : Ptr. auf den aufgefüllten String
*/
char *strpad(char *s, int len)
{
  strfill(s + strlen(s), s + len - 1, ' ');
  *(s + len) = '\Ø';
  return (s);
}
/*============================================================

strcompress
-----------
Funktion: Umkehrung von STRPAD. Entfernt alle "rechtsbündigen"
          Leerzeichen. Komprimiert einen String auf seine
          "echte" Länge durch entsprechendes Setzen von '\Ø'.
Hin:      s      : Ptr. auf den String
         RETURN  : Ptr. auf den String
*/
char *strcompress(char *s)
{
  char *p;

  p = s;
  s += strlen(s) - 1;
  while (s >= p  &&  *s == ' ')
     s--;
  *(++s) = '\Ø';
  return (p);
}
/*============================================================
```

Listing 2: *(Fortsetzung)*

```
lower
-----
Funktion: Wandelt alle Großbuchstaben eines Strings in
          Kleinbuchstaben
Hin:      s        : Ptr. auf String
         RETURN    : Ptr. auf String
*/
char *lower(char *s)
{
  char *p;

  p = s;
  while (*s) {
     switch (*s) {
        case 'Ö' : *s = 'ö'; break;
        case 'Ä' : *s = 'ä'; break;
        case 'Ü' : *s = 'ü'; break;
        default  : *s = tolower(*s);
     }
     s++;
  }
  return(p);
}
/*============================================================

upper
-----
Funktion: Wandelt alle Kleinbuchstaben eines Strings in
          Großbuchstaben
Hin:      s        : Ptr. auf String
         RETURN    : Ptr. auf String
*/
char *upper(char *s)
{
  char *p;

  p = s;
  while (*s) {
     switch (*s) {
        case 'ö' : *s = 'Ö'; break;
        case 'ä' : *s = 'Ä'; break;
        case 'ü' : *s = 'Ü'; break;
        default  : *s = toupper(*s);
     }
     s++;
  }
  return(p);
}
/*============================================================

instr
-----
Funktion: Prüft, ob der String <s1> im String <s2> enthalten
ist
Hin:      s1       : Ptr. auf den Vergleichsstring
          s2       : Ptr. auf den durchsuchten String
         RETURN    : Zeiger auf gefundene Position
                     (NULL=nicht enthalten)
*/
char *instr(char *s1, char *s2)
{
  char *ptr1, *ptr2;

  for ( ; *s2; s2++) {
     for (ptr1 = s1, ptr2 = s2;
        *ptr1 && (*ptr1 == *ptr2) ;
        ptr1++, ptr2++)
        ;
     if (! *ptr1)
        return (s2);
  }
  return (NULL);
}
/*============================================================
```

Listing 2: *(Fortsetzung)*

```
nextword
--------
Funktion: Sucht in einem String das nächste Wort (nach rechts)
Hin:        s    : Ptr. auf String
            term : Ptr. auf String mit Trennzeichen zwischen
                   Wörtern
         RETURN  : Ptr. auf Wortanfang (NULL, wenn kein weiteres
                   Wort)
*/
char *nextword(char *s, char *term)
{
   while (*s  &&  (! strchr(term, *s)))
      s++;
   while (*s  &&  (strchr(term, *s)))
      s++;
   return (*s  ?  s  :  NULL);
}
/*==============================================================

prevword
--------
Funktion: Umkehrung von NEXTWORD. Sucht - ausgehend von der
          Position <ende> - das vorhergehende Wort (nach
links).
          Die Suche endet, wenn die Position <start> erreicht
          ist (normalerweise wird als <start> der Stringanfang
          übergeben. Kriterium für "Wort-Terminatoren" ist
          wieder der Inhalt von TERM.
Hin:      start : Ptr. auf die linke Grenze
          ende  : Ptr. auf die rechte Grenze
          term  : Ptr. auf String mit Trennzeichen zwischen
                  Wörtern
        RETURN  : Ptr. auf Wortanfang (NULL, wenn kein letztes
                  Wort)
*/
char *prevword(char *start, char *ende, char *term)
{
   while ((start <= ende)  &&  (! strchr(term, *ende)))
      ende--;
   while ((start <= ende)  &&  (strchr(term, *ende)))
      ende--;
   while ((start <= ende)  &&  (! strchr(term, *ende)))
      ende--;
   return (*ende  ?  ende + 1  :  NULL);
}
/*==============================================================

stradd
------
Funktion: Verknüpft eine variable Anzahl Strings durch
          "Konkatenieren" (wiederholter Aufruf von STRCAT) zu
          einem Gesamtstring.
          Resultierende Länge: strlen(arg1) + ... +
strlen(argN)
Hin:        s    : Ptr. auf den "Summenstring"
          arg1 : Ptr. auf 1.Teilstring
          arg2 : Ptr. auf 2.Teilstring
          ...
          ...
          argN : Ptr. auf N.Teilstring
          NULL : Endemarke
       RETURN :  Ptr. auf erzeugten "Summenstring"
          Bsp.: stradd(s,"Dies ","ist ","ein ","Test", NULL);
*/
char *stradd(char *s, ...)
{
   char **argp;

   argp = &s;
   argp++;
   strcpy(s, *argp++);
   while (*argp) {
      strcat(s, *argp);
      argp++;
   }
   return (s);
}
/*==============================================================
```

Listing 2: *(Fortsetzung)*

```
charadd
-------
Funktion: Verknüpft eine variable Anzahl Zeichen zu einem mit
          '\0' abgeschlossenen String.
          Resultierende Länge: Anzahl Zeichen
Hin:        s    : Ptr. auf den "Summenstring"
          arg1 : 1.Zeichen
          arg2 : 2.Zeichen
          ...
          ...
          argN : N.Zeichen
          '\0' : Endemarke
       RETURN :  Ptr. auf erzeugten String
          Bsp.: charadd(s, 'T', 'e', 's', 't', '\0');
*/
char *charadd(char *s, ...)
{
   char *argp, *p;

   p = s;
   argp = &s;
   for (argp += sizeof(char *); *s = *argp; s++, argp += 2);
   return (p);
}
```

Listing 2: `MYSTRING.C`.